PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

ESCUELA DE SISTEMAS

DISERTACIÓN PREVIA A LA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN SISTEMAS Y COMPUTACIÓN

# "DISEÑO Y DESARROLLO DE UNA INTERFACE EN DOBLE VÍA DE UNA EMPRESA DE VENTA DE CONSUMO MASIVO / OPERADOR LOGÍSTICO Y VICEVERSA"

JOHN PAUL TOBAR ESTRELLA

DIRECTOR: OSWALDO ESPINOSA

QUITO, 2016

Agradecimiento

Agradezco a mi director de tesis Oswaldo Espinosa por su colaboración y apoyo en la elaboración de este trabajo.

Agradezco a mis hermanos por el respaldo constante e incondicional para ayudarme a crecer personal y profesionalmente.

Agradezco a mi tío Pablo Estrella por sus consejos y guías que me llevaron a escoger esta excelente profesión.

Dedicatoria

Dedico esta disertación a mis padres que han sido un ejemplo de lucha y mejoramiento continuo.

# Resumen

El objetivo de esta disertación es el diseño y desarrollo de una interface en doble vía de una empresa de consumo masivo con el operador logístico, mediante la automatización del proceso de verificación de inventario para la facturación, dando como beneficios mantener diariamente un inventario cuadrado y la disminución en el tiempo de facturación.

En el primer capítulo se aborda el marco general del país, los objetivos y se describe los requerimientos de los usuarios con los cuales se ha desarrollado esta disertación.

En el segundo capítulo se abordan los fundamentos teóricos con la bibliografía pertinente, y se explica el lenguaje de programación con el cual se desarrolla el sistema.

En el tercer capítulo se analizan los procesos operativos de la empresa entre ellos el manejo de los clientes, el crecimiento de la empresa, así como la problemática que ha representado dicho crecimiento, además en este capítulo se propone una solución a dicha problemática.

El cuarto capítulo está dividido en cuatro subtemas los cuales son: i) análisis de requisitos mediante la especificación de los requerimientos de usuarios, ii) diseño del sistema mediante diagramas de secuencia, casos de uso, diagramas de base de datos, iii) definición de pruebas a realizar para la verificación del correcto funcionamiento del sistema, y por último, iv) manuales de instalación y de usuario a modo de anexo.

En el quinto capítulo se presentan las conclusiones y recomendaciones.

Por último el capítulo seis hace referencia a la bibliografía utilizada en esta disertación.

# Contenido

# Índice de Ilustraciones

## Índice de Tablas

# Capítulo 1: Introducción

## Marco general del país

El desarrollo industrial ha revolucionado la forma de vivir de las personas en general, más allá del lugar de residencia de un individuo, o sus costumbres o clase social. Es por esto que cada día aumenta los requerimientos de satisfacer sus necesidades primarias y secundarias, es así que en esta demanda constante de productos y servicios las empresas en general han visto una oportunidad de crecimiento y desarrollo de productos.

Entre las empresas que se han desarrollado en la capacidad de satisfacer la demanda se encuentran empresas multinacionales de consumo masivo. Estas empresas si bien tienen una alta capacidad de producción que le favorecen a su rentabilidad, también tienen que lidiar con las problemáticas que conlleva en las diferentes etapas de producción, comercialización, ventas y almacenaje al tener varias macrocategorías[1].

Es así que esta tesis busca solucionar la problemática específica de una empresa de consumo masivo dedicada a la producción, comercialización y almacenaje de productos de consumo masivo.

Dentro de los procesos operativos vitales de la empresa se encuentra el almacenaje de estos productos.

La empresa dado su acelerado crecimiento de desarrollo en el mercado ecuatoriano ha identificado una problemática de almacenamiento, ya que todos sus productos no

[1] Macrocategorías son las agrupaciones de productos como por ejemplo: bebidas, cuidado personal, etc.

se pueden almacenar en las bodegas de la empresa, debido a la limitación de espacio. Por lo que se contrató a un operador logístico [2]para almacenar y alistar el inventario.

Esta logística consistía en una comunicación constante, manual y poco tecnificada entre las dos empresas, por lo que lo hacía un proceso lento y poco eficiente en el almacenaje y alistamiento del inventario, debido al retraso en la ejecución de estos procesos.

## 1.1 Objetivos

### 1.1.1 Objetivo general

- Diseñar y desarrollar una interface en doble vía que permita solicitar productos y recibir la confirmación del producto disponible.

### 1.1.2 Objetivos específicos

- Realizar el levantamiento de requerimientos para conseguir una interface, que incluya transferencia automática de archivos en doble vía entre el Operador Logístico y la empresa de venta de consumo masivo.
- Identificar los requerimientos funcionales y no funcionales de la aplicación.

[2] Operadores logísticos son las empresas que se encargan de almacenaje, alistamiento, distribución, venta de producto, etc.

- Realizar el diseño lógico y físico de la aplicación a automatizar, considerando las características, restricciones de la operación y del sistema.
- Realizar el diseño físico de la aplicación con los entregables que se generarán.
- Desarrollar la aplicación utilizando las herramientas de desarrollo seleccionadas.
- Realizar los casos de prueba en base a los casos de uso.
- Realizar el manual del usuario.

## 1.2 Descripción de los requerimientos

Hoy en día el intercambio de información con el operador logístico se realiza a través de hojas de Excel, las cuales son digitadas manualmente, por lo que se genera grandes oportunidades en la creación de las mismas.

La empresa requiere modificar sus procesos y agregar a su ERP[3] un sistema con el cual se logre automatizar el intercambio de información con el operador logístico, permitiendo solicitar la verificación de productos según la cantidad que se vaya a facturar diariamente y así como el alistamiento de productos según sus políticas de despacho, adicionalmente se requiere hacer un seguimiento diario a los movimientos del inventario por parte del operador.

[3] ERP(Enterprise Resource Planning) sistema que permite integrar todos los procesos vitales de una empresa.

# Capítulo 2: Fundamentos teóricos

## 2.1 Metodologías

Las metodologías de desarrollo de software es un conjunto de pasos o técnicas que nos permite definir, planear, codificar, probar el software deseado según las necesidades de los usuarios.

Para la creación de software se necesita definir una gran cantidad de tareas y etapas, por lo que se observó la necesidad de crear unas técnicas o guías que permitan definir, planificar, desarrollar, probar, implementar y mejorar el trabajo a realizar, para de esta manera entregar un software de calidad, cumpliendo con las necesidades del usuario.

Teniendo en cuenta que el software es distinto, se ha creado dos tipos de metodologías, metodologías tradicionales o modernas que nos permiten realizar el desarrollo de software de una manera ordenada documentada o de una manera más ágil, susceptible a cambios. Para seleccionar la metodología que se va a usar se debe tomar en cuenta las ventajas y desventajas de cada una, así como el entorno y que tan cambiante es el mismo.

### 2.1.1 Metodologías Tradicionales

Las metodologías tradicionales están basadas en la planificación, documentación, disciplina y definición del proyecto.

La filosofía de las metodologías tradicionales se basa en definir detalladamente las fases iniciales y tareas de un proyecto, así como las herramientas a utilizar, como resultado se obtendrá un gran volumen de documentación.

El objetivo de estas metodologías es prever todas las necesidades y todos los eventos externos, para obtener todos los insumos necesarios para definir un correcto plan de trabajo, el cual se deberá cumplir de una manera muy rigurosa.

Estas metodologías son más eficaces y ventajosas cuando se va a desarrollar un proyecto de gran magnitud, para aplicarla se requiere una organización sólida y bien definida, ya que se necesitará del apoyo y experticia de usuarios, ya que deben conocer profundamente los requerimientos del sistema.

Ente las metodologías tradicionales tenemos:

#### *2.1.1.1 Modelo en espiral*

Otro modelo es el espiral propuesto por Barry Boehm de tipo evolutivo que consta de una serie de ciclos divididos en cuatro tareas. Cuando se han cumplido los objetivos de un ciclo se pasa al siguiente, ver ilustración 2.1

Las tareas de cada ciclo son:

– En la primera fase de cada iteración se determinan los objetivos para esa misma iteración, se proponen alternativas y se detectan las posibles restricciones.

– En el análisis de riesgos se decide si, para las alternativas elegidas, vale la pena seguir con el proyecto en esa iteración o abandonarlo definitivamente.
– El desarrollo en las primeras iteraciones consiste en un prototipado, mientras que en las últimas iteraciones se desarrollará el sistema completo con todos los detalles. Al final de esta fase se deben hacer pruebas para comprobar que el sistema cumple con los requisitos programados.
– A continuación se hace una evaluación del producto obtenido en esa fase y con los resultados obtenidos se planifica la siguiente iteración.

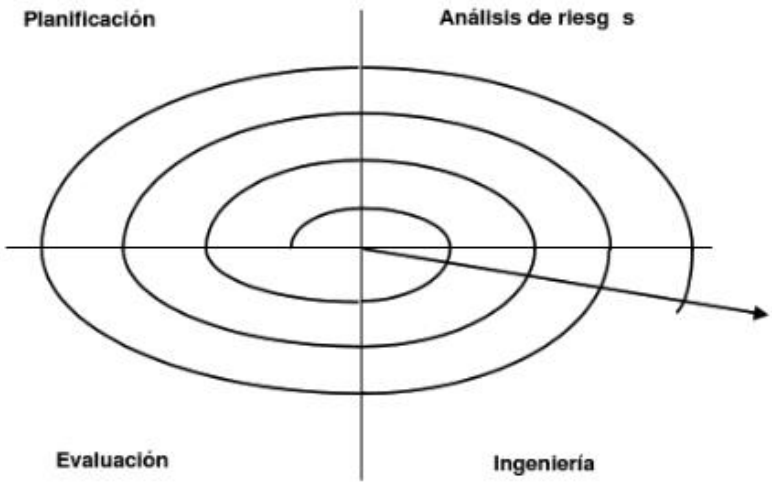


*Ilustración 0.1 Modelo en espiral (Teniente López, Costal Costa, & Sancho Samsó, 2013)*

El modelo en espiral tiene la ventaja de que el sistema se puede evaluar mucho antes que con el modelo en cascada. Al final de cada iteración se tiene una versión del producto; por lo tanto es un modelo que responde mejor ante variaciones del diseño original. Además este modelo introduce nuevos conceptos, como la creación de prototipos y el análisis de riesgos. (Lajara Vizcaíno, 2007)

#### *2.1.1.2 Modelo en V*

El modelo en V es una evolución del modelo de ciclo de vida en cascada. En él, las etapas se organizan en una estructura en forma de V. En el lado izquierdo se representa la descomposición de las necesidades: requisitos y especificación del sistema, diseño, etc. Mientras que en el lado derecho se muestra la integración de las piezas y la verificación del sistema.

Este modelo contiene las etapas clásicas del desarrollo: una fase de análisis, una fase de diseño, programación y verificación. Sin embargo, existe una conexión directa entre las etapas de pruebas y las de desarrollo. Cualquier error de diseño detectado en la etapa de pruebas conduce al rediseño y nueva programación del código afectado. Las etapas de desarrollo y diseño deben realizarse de forma paralela con las actividades de pruebas y existir una comunicación activa entre los técnicos de pruebas, analistas y desarrolladores. La principal ventaja de este modelo es que la relación entre las etapas de desarrollo y de pruebas facilita la localización de errores. Es un modelo sencillo, en el que se especifican correctamente los roles de las distintas pruebas a realizar y además involucra de forma activa al usuario en ellas. Entre sus desventajas, destaca que el cliente no forma parte activa del proyecto durante su desarrollo, que el producto final puede no reflejar todos los requisitos del usuario y, además, que las pruebas pueden ser costosas y no lo suficientemente efectivas. (Villada Romero, 2015)

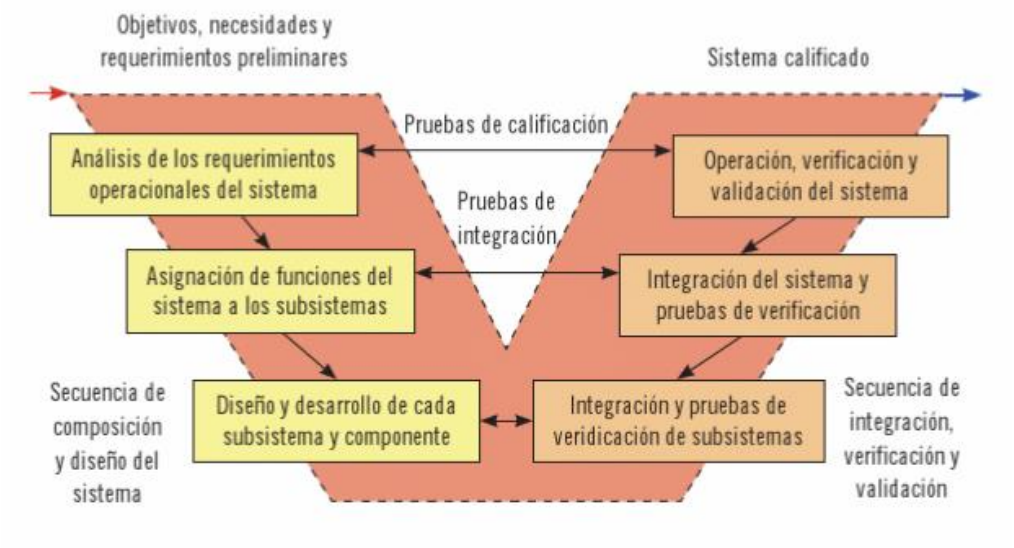


*Ilustración 0.2 Modelo en V (Villada Romero, 2015)*

La unión de las líneas entre los dos lados ayuda a identificar el tipo de pruebas que se deben realizar y en que parte se debe corregir si se encuentra un error.

El modelo tiene cuatro niveles, en cada nivel existe una fase de definición y su fase de pruebas para la verificación que satisfaga los requerimientos de los usuarios.

Los niveles son los siguientes:

- Nivel 1: Este nivel está enfocado en las necesidades del cliente. El cliente debe definir sus requerimientos y verificar que se cumplan sus necesidades.
- Nivel 2: Este nivel está enfocado en la definición de las funcionalidades del sistema.
- Nivel 3: Este nivel está enfocado en definir los elementos técnicos que tiene el sistema, esta fase se la denomina Arquitectura del Sistema.
- Nivel 4: Este nivel está enfocado en desarrollar los requerimientos definidos por el cliente.

### *2.1.1.3 Rational Unified Process (RUP)*

El RUP nace en el año 1998 a partir de las aportaciones de G. Booch y J. Rumbaugh al método de desarrollo Objectory creado por I. Jacobson. Más que un único método de desarrollo, el RUP pretende ser un marco de trabajo general que se pueda especializar según las necesidades de cada empresa.

El RUP organiza el desarrollo del proyecto en dos dimensiones (véase la ilustración 2.3). El eje horizontal representa el tiempo y muestra el aspecto dinámico del proceso de desarrollo. El eje vertical representa el aspecto estático y muestra las diferentes actividades que se realizan en cada instante de tiempo. (Cabot Sagrera, 2013)

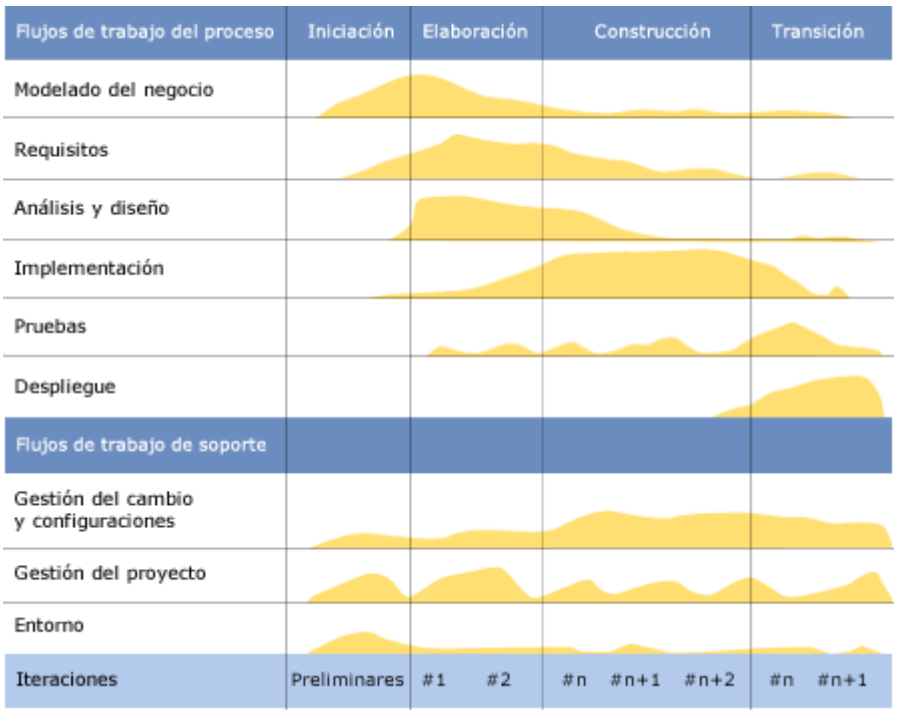


*Ilustración 0.3 RUP(Cabot Sagrera, 2013)*

Para pasar a las siguientes fases se tiene una meta, la cual permite decidir si se continúa con la siguiente fase, en cada fase se van a realizar una o más iteraciones para ir cumpliendo las metas en cada una, generalmente en la fase de construcción es donde se realizan el mayor número de iteraciones.

RUP se divide en cuatro fases: inicio, elaboración, construcción y transición.

- Fase Inicio: en esta fase se identifican los actores que van a actuar con el sistema y cuál será la interacción de cada uno de ellos. Al final se debe tener una idea clara de los requerimientos del sistema por parte de los usuarios.
- Fase Elaboración: en esta fase se perfeccionan los requerimientos del usuario y se define la arquitectura del sistema en base a los casos de uso más críticos, se crea un prototipo y poder identificar los riesgos.
- Fase Construcción: en esta fase se desarrolla todos los casos que se definieron y se prueban todas las funcionalidades del sistema.
- Fase Transición: en esta fase se implementa el sistema para que los usuarios puedan capacitarse, completar la documentación, etc.

Las actividades según este método (véase la ilustración 2.4) son la recogida de requisitos, el análisis, el diseño, la implementación y las pruebas. Aparte de éstas, RUP también define las tareas de apoyo siguientes: configuración y gestión de los cambios, gestión del proyecto y gestión del entorno, que pretenden ayudar a asegurar la calidad del software y minimizar los riesgos durante su construcción, dos de los objetivos más importantes dentro de la ingeniería del software. (Cabot Sagrera, 2013)

#### *2.1.1.4 Modelo iterativo*

El modelo anterior es la primera aproximación que se da para acometer el proceso de desarrollo de un proyecto. Sin embargo, pronto aparecen los primeros problemas que derivan de este modelo. En concreto, en el modelo en cascada se llega a un acuerdo con el cliente en la primera fase del proyecto

sobre la funcionalidad del sistema y no existe ningún vínculo con este hasta la finalización del sistema. Es decir, no existe retroalimentación por parte del cliente. Surge entonces el modelo iterativo, para minimizar el riesgo que supone no contar con la opinión del cliente durante todo el desarrollo del sistema. Este modelo se compone de iteraciones, donde en cada iteración se produce la secuencia de etapas de un modelo en cascada clásico. Una iteración es, por tanto, un conjunto de períodos de tiempo donde se produce una versión ejecutable del producto y la documentación necesaria. Cada iteración posee una fase de análisis para determinar cuál es la mejora que se va a realizar sobre el sistema en esa iteración y existe una fase de entrega del módulo elaborado.

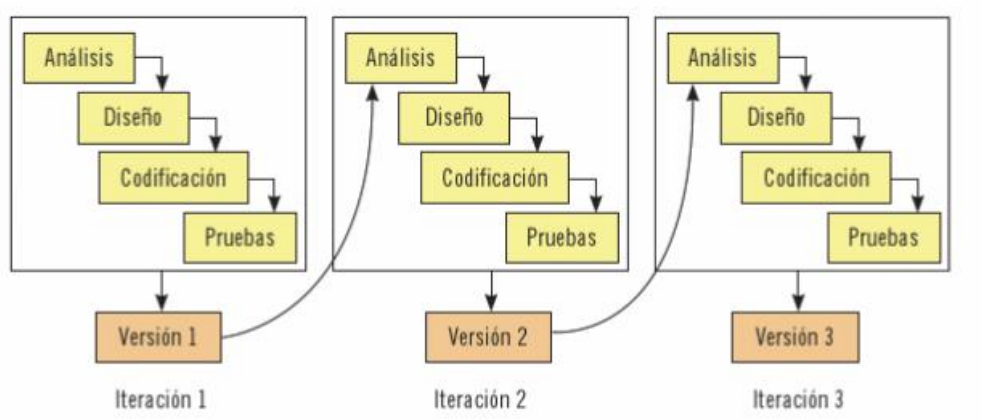


*Ilustración 0.4 Secuencias de etapas del proceso iterativo (Villada Romero, 2015)*

De esta manera, los objetivos de cada iteración se establecen en función de la evaluación por parte del cliente de las anteriores, corrigiéndolas o proponiendo mejoras.

Este modelo permite entre otras cosas disminuir los riesgos que supone un cambio en los requisitos del sistema durante el desarrollo, reduciendo costes y proponiendo una participación activa del cliente dentro del proyecto. (Villada Romero, 2015)

#### *2.1.1.5 Modelo incremental*

El modelo incremental se centra en desarrollar el sistema en partes, de forma que se van entregando a medida que se van completando. Este modelo se adapta mejor a sistemas con una gran complejidad funcional. A veces, el modelo incremental se confunde con el iterativo y viceversa. La diferencia principal es que el modelo iterativo produce en cada iteración una versión mejorada de la anterior iteración, mientras que el modelo incremental añade en cada iteración una parte de la funcionalidad del sistema. Por lo tanto, en cada iteración se obtiene una versión estable del sistema.

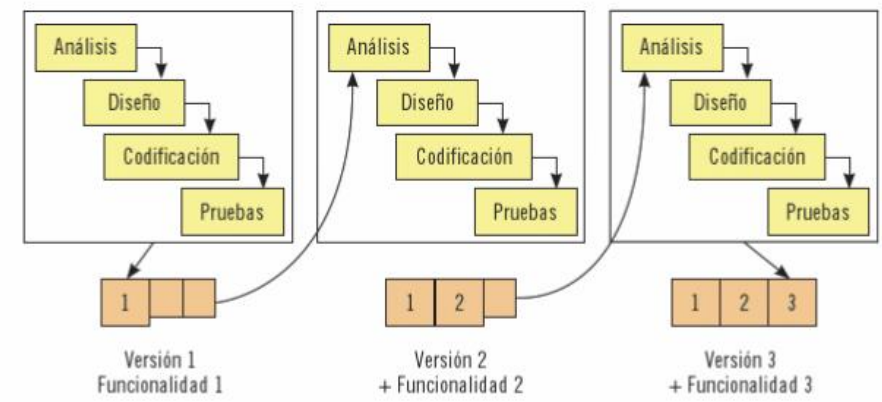


*Ilustración 0.5 Secuencia de etapas del proceso incremental (Villada Romero, 2015)*

Las dos claves de este modelo son: comenzar con una implementación simple de los requisitos del sistema e ir mejorando y añadiendo nueva funcionalidad hasta que el sistema esté completo, y, por otro lado, usar la experiencia y conocimientos adquiridos en anteriores iteraciones. (Villada Romero, 2015)

### 2.1.2 Metodologías Ágiles

A finales de 1990's, varias metodologías comenzaron a llamar la atención del público, cada metodología tenía su propia combinación de nuevas y viejas ideas. Dichas metodologías hicieron énfasis en la colaboración cercana entre el equipo de desarrollo y los clientes; frecuentes entregables del trabajo desarrollado, correcciones y las verificaciones de los clientes.

El término "Ágil" se aplicó a este grupo de metodologías a inicios de 2001, cuando diecisiete expertos desarrolladores se reunieron en Snowbird, Utah, en esa reunión discutieron sus ideas y los enfoques para el desarrollo de software. Se esta reunión nacieron los valores del desarrollo ágil y sus principios, los cuales fueron plasmados en el Manifiesto Ágil para el desarrollo ágil de software.

Tras esta reunión se formó The Agile Alliance, una organización sin ánimo de lucro, para fomentar a los desarrolladores que exploren, compartan, discutan sus ideas y experiencias con el desarrollo ágil.

El Manifiesto Ágil

Se ha descubierto nuevas y mejores formas de desarrollar software. A través del tiempo se ha llegado a valorar cuatro principios, los cuales son:

1. Individuo e interacciones del equipo sobre el proceso y las herramientas.
2. Trabajar en el software en lugar de realizar una amplia documentación.
3. Colaboración con el cliente sobre la negociación de contratos.
4. Responder a los cambios en lugar de seguir un plan.

(Agile Alliance, 2016)

Los principios se basan más en el bienestar de las personas y los requisitos cambiantes, dando una mayor importancia al trabajo con los clientes y satisfacer

sus necesidades. Los valores antes mencionados han inspirado los doce principios del manifiesto, estos valores son las principales características que diferencian las metodologías tradicionales con las ágiles.

Los principios del manifiesto ágil son:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante
   de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.

10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

(Agile Manifesto, 2016)

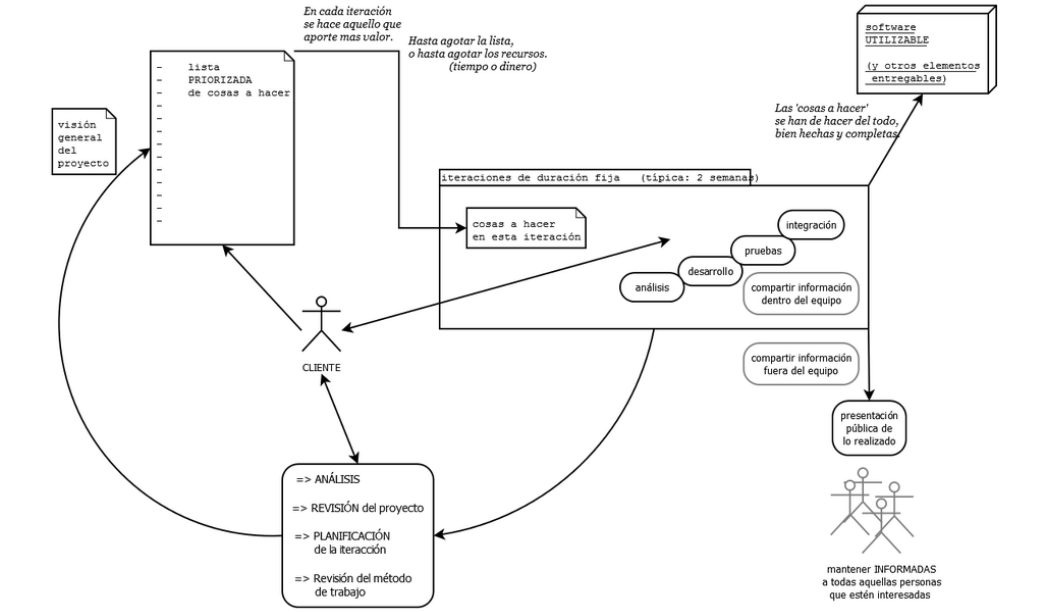


*Ilustración 0.6 Esquema General Metodologías ágiles (Wikipedia, 2016)*

Entre las metodologías agiles tenemos:

### *2.1.2.1 XP (eXtreme Programming)*

La programación extrema (XP-eXtreme Programming) es una metodología ágil para el desarrollo de proyectos de software que realiza un plan de proyecto basado en versiones del producto acordadas a partir de funcionalidades

concretas. Una vez entregada la versión del proyecto y al cumplir con los requisitos, el proceso vuelve a iniciarse con un conjunto mayor de funcionalidades. (López Goytia, 2014)

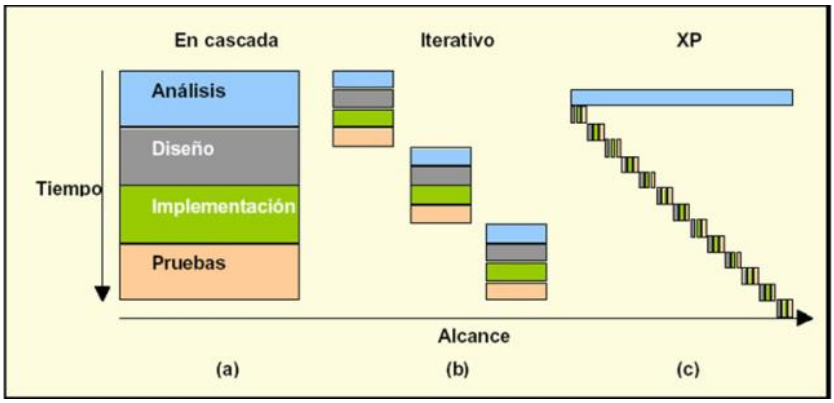


*Ilustración 0.7 Programación extrema (Osl2, 2016)*

En la ilustración 2.7 se puede observar la evolución de los largos ciclos de desarrollo en cascada (a), ciclos iterativos más cortos (b) y la mezcla que hace XP (c).

Entre las metodologías ligeras, la programación extrema es una de las que más se están extendiendo, tratando de evitar la anarquía de este tipo de situaciones. Se basa fundamentalmente en dividir el proyecto en hitos muy cortos de desarrollo, habitualmente de una semana. Los requisitos se toman semanalmente, basándose en ideas simples que todo el mundo pueda entender y en las que puedan centrarse, por lo que la aplicación se construye con realimentación inmediata del cliente. Nos muestra que para alcanzar el éxito en un proyecto se deben cumplir un conjunto de prácticas equilibradamente:

– Cliente in situ. El cliente debe estar siempre disponible, preferiblemente como parte del equipo. Si el cliente no tiene los mismos incentivos y motivación que el grupo de desarrollo para la ejecución del proyecto, éste no prosperará.

- Juego de planificación. Consiste en la negociación del alcance del proyecto para una interacción entre cliente y equipo de desarrollo. Lo ideal es que el cliente marque el alcance de negocio y los desarrolladores el alcance técnico.
- Metáforas del sistema. Es una forma conceptual de poner un nombre (una frase resumen) a los diferentes elementos del sistema, basándose en una metáfora conocida por el equipo.
- Relatos del usuario. Son una característica que el cliente desea que sea incluida en el producto a modo de requisitos entendibles por el equipo de desarrollo. Normalmente toma la forma de un breve relato de lo que debe hacer el sistema desde el punto de vista del usuario.
- Cuarenta horas semanales. El proyecto se divide en fases de cuarenta horas semanales. El riesgo asumido en cada fase es pequeño, y se pueden detectar rápidamente las desviaciones del calendario. También consideramos que una persona no debe trabajar más horas de las debidas para que no se llegue al agotamiento y baje la motivación.
- Re-codificación. Cuando se detectan trozos de código susceptibles de ser mejorados, se reescriben o reorganizan.
- Diseño sencillo. Dado que los desarrollos son muy cortos, los diseños deben ser muy sencillos. No debemos pensar cómo será nuestro sistema dentro de un año, sino dentro de un mes.
- Programación por parejas. Un desarrollador programa y el otro revisa y prueba. Se aplica el lema cuatro ojos ven más que dos. Se pueden ir alternando los papeles. Bien entendido puede ser muy útil matizándolo un poquito: un programador experto, que podría haber sido subcontratado para

esa labor, puede asesorar a programadores menos experimentados para garantizar un buen rendimiento.

- Pruebas unitarias. Cada elemento que se construye dispone de un conjunto de pruebas unitarias que, al ejecutarse, comprueban su corrección. Por ejemplo, si hacemos un componente responsable de almacenar un objeto en una base de datos, le adjuntamos unos programas de pruebas unitarias que lo insertan, borran y modifican. El tiempo empleado en pruebas es siempre más amplio de lo que pensamos. Si nos acostumbramos a automatizarlas pronto, se convertirán en una pieza imprescindible de nuestra dinámica diaria.
- Versiones cortas. El sistema de versionado del producto se basa en versiones muy cortas en tiempo de desarrollo, que incluyen paquetes pequeños de funcionalidad nueva. Así también se minimiza el riesgo.
- Propiedad colectiva. Todo el mundo contribuye en todas partes del proyecto. Cuando encuentra un bug[4], lo corrige, o añade nueva funcionalidad. No se crean cuellos de botella ni dependencias. Se trata de evitar situaciones del estilo "Esto no es mío y Carlos ya lo arreglará cuando venga de vacaciones". Las porciones de proyectos son de las organizaciones no de los individuos.
- Estándares de codificación. Se sigue un conjunto de estándares de codificación previamente definidos. Por ejemplo, en los desarrollos Java ya hay un conjunto de estándares de estilo de código fuente que las herramientas de desarrollo respetan.

[4] Un bug es un error o un defecto en el software o hardware que hace que un programa funcione incorrectamente.

– Integración continúa. Los diferentes equipos de desarrollo integrando sus desarrollos van por turnos en el repositorio principal. (Canales Mora, 2010)

### *2.1.2.2 AUP (Agile Unified Process)*

AUP es una versión simplificada del RUP. En él se describe un enfoque simple, fácil de entender para el desarrollo de software usando técnicas y conceptos ágiles; y aun así se mantiene la idea de RUP.

En la ilustración 2.9 se puede observar que el eje horizontal no ha cambiado, el objetivo en cada fase es igual que en RUP, mientras que el eje vertical ha cambiado. En primer lugar, la actividad Modelo abarca las actividades del modelado del negocio, requisitos y análisis y diseño. Modelo es una actividad importante del AUP, pero no domina el proceso. En segundo lugar, la actividad gestión del cambio y configuración es ahora la Gestión de la configuración.

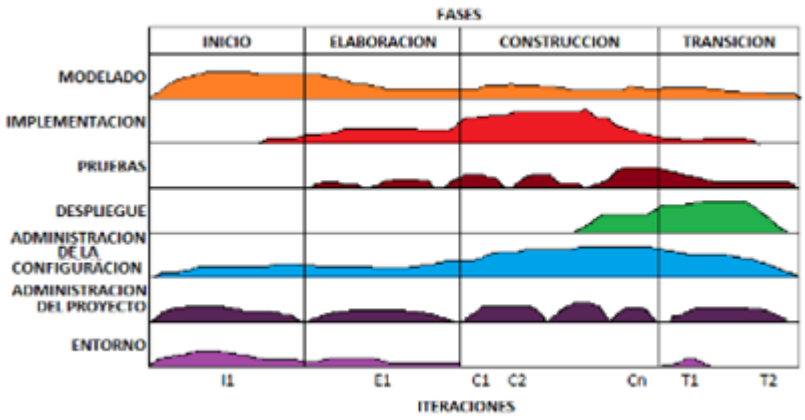


*Ilustración 0.8 AUP (Ubuntu Adempiere, 2016)*

Las actividades que se realizan son:

– Modelo. El objetivo de esta actividad es entender la organización del negocio, el problema que se desea solucionar con el proyecto, e identificar una solución viable para hacer frente al problema.

- Implementación. El objetivo de esta actividad es transformar el modelo en código ejecutable, para llevar a cabo un nivel básico de las pruebas, en particular las pruebas unitarias.
- Prueba. El objetivo de esta actividad es llevar a cabo una evaluación objetiva para garantizar la calidad del software. Esto incluye la búsqueda de defectos, la validación de que el sistema funciona tal como fue diseñado y verificar que se cumplen los requisitos.
- Despliegue. El objetivo de esta actividad es la entrega del software a los usuarios finales.
- Gestión de la Configuración. El objetivo de esta actividad es gestionar el acceso a los elementos del proyecto. Esto no solo incluye el seguimiento de las versiones, también incluye el control y gestión de los cambios presentados.
- Gestión de proyectos. El objetivo de esta actividad es gestionar las tareas que se realizan en el proyecto.
- Ambiente. El objetivo de esta disciplina es asegurar que todos los elementos estén disponibles y sean los adecuados para el equipo.

El AUP se basa en los siguientes principios:

1. Su personal sabe lo que están haciendo. La gente no va a leer la documentación detallada del proceso, pero ellos van a querer alguna orientación de alto nivel y/o la formación de vez en cuando.
2. Sencillez. Todo se describe de forma concisa utilizando un puñado de páginas, no miles de ellos.

3. Centrarse en actividades de alto valor. La atención se centra en las actividades que en realidad cuentan.

(Ambysoft, 2016)

#### *2.1.2.3 Scrum*

Scrum es un framework de desarrollo de proyectos que ayuda a responder de una manera ágil a los cambios existentes en el desarrollo del proyecto. El equipo principal de SCRUM consta de 3 roles:

- SCRUM Master, es un líder servicial, quien se asegura que todos los miembros del equipo cumplan estrictamente la metodología SCRUM y elimina los impedimentos para el correcto desarrollo del proyecto.
- Product Owner, es el puente entre los usuarios, clientes y el equipo SCRUM, es la voz del cliente, es la persona quien va a aceptar o rechazar los entregables desarrollados por el equipo SCRUM.
- Equipo SCRUM, son los miembros del equipo, los cuales deben ser multidisciplinarios, auto-organizados, contributivos, responsables, ya que el avance del proyecto dependerá del trabajo realizado por este equipo.

Los elementos básicos de SCRUM son:

- Product Backlog: es un listado con las funcionalidades de la aplicación, esta lista es definida por el Product Owner.
- Sprint Backlog: en cada Sprint se van a obtener las historias de usuario basadas del Product Backlog, las cuales deberán ser desarrolladas en el actual Sprint.

- Tablero SCRUM: es un tablero en donde se colocan las tareas que se realizan a lo largo del proyecto, por lo general se definen 3 estados, por hacer, haciendo, hechos. Las tareas pueden cambiar de estado únicamente si terminan por completo las actividades de dicha tarea.

Además de estos elementos tenemos unas cuantas reglas básicas y sencillas que tenemos que cumplir, las cuales son:

- Una vez se comience el Sprint no se pueden cambiar las tareas definidas para el actual Sprint.
- Al final de todo Sprint el equipo SCRUM debe desarrollar un entregable.
- Durante el Sprint a diario se realiza la reunión Stand-up, en la cual participan todos los integrantes del equipo Scrum y cada uno debe responder a las preguntas Que hice ayer? Que voy hacer hoy? Que problemas he tenido? La duración de la reunión no puede ser mayor a 15 minutos.
- Al final de cada Sprint, el equipo Scrum presenta los entregables al Product Owner, quien acepta o rechaza los entregables
- La duración del "Sprint" se define al inicio del proyecto y no se puede modificar a lo largo del proyecto. Para definir el tiempo de un Sprint se debe tomar en cuenta la estabilidad del entorno del proyecto, cuando existe un entorno estable se puede definir Sprint largos, si existe un entorno muy cambiante se debe definir Sprint cortos.
- La única persona que puede detener un Sprint en curso es el Product Owner.

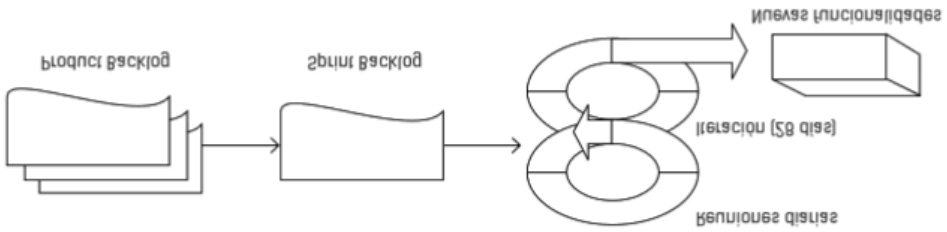

*Ilustración 0.9 Proceso básico SCRUM (López Goytia, 2014)*

### *2.1.2.4 Diferencias entre metodologías*

La aplicabilidad de las metodologías va a depender con el proyecto que se desarrolle, por lo que no se puede definir a una sola metodología como la óptima, cada metodología sea ágil o tradicional tienen sus ventajas y sus desventajas. Según la experticia del equipo de desarrollo se podrá definir la mejor metodología que se va a usar, según el tipo de proyecto. Para seleccionar una metodología se debe identificar correctamente el ambiente, el equipo de usuarios, los posibles cambios que se presenten en el desarrollo del proyecto, etc.

En términos generales podemos decir que las metodologías tradicionales se centran en una buena documentación de todo el ciclo de vida, mientras las metodologías ágiles están enfocadas en tener una excelente capacidad de respuesta a los cambios que se presenten en el desarrollo del proyecto.

A continuación se tiene una tabla con la comparación de metodologías:

*Tabla 1 Comparación metodologías (Metodologías Agiles, 2016)*

| **Metodologías Agiles** | **Metodologías Tradicionales** |
|---|---|
| • Basadas en heurísticas | • Basadas en normas provenientes de |

| | |
|---|---|
| provenientes de prácticas de producción de código<br>• Especialmente preparados para cambios durante el proyecto<br>• Impuestas internamente (por el equipo)<br>• Proceso menos controlado, con pocos principios<br>• No existe contrato tradicional o al menos es bastante flexible<br>• El cliente es parte del equipo de desarrollo<br>• Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio<br>• Pocos artefactos<br>• Pocos roles<br>• Menos énfasis en la arquitectura del software | estándares seguidos por el entorno de desarrollo<br>• Cierta resistencia a los cambios<br>• Impuestas externamente<br>• Proceso mucho más controlado, con numerosas políticas/normas<br>• Existe un contrato prefijado<br>• El cliente interactúa con el equipo de desarrollo mediante reuniones<br>• Grupos grandes y posiblemente distribuidos<br>• Más artefactos<br>• Más roles<br>• La arquitectura del software es esencial y se expresa mediante modelos |

### 2.1.3 Detalle de la metodología de cascada seleccionada.

La metodología de Cascada también es conocida como metodología clásica, tradicional o lineal.

La característica principal de esta metodología se basa en seguir rigurosamente las fases del ciclo de vida del software, por lo tanto no podremos avanzar de fase mientras no finalicemos la etapa en la que nos encontremos, en caso de encontrar un error en una fase se debe permanecer en esa fase hasta que esta se encuentre lista.

Al inicio del proyecto se debe asegurar que están bien definidos todos los requisitos de los usuarios para poder realizar un correcto diseño de la solución, el asegurar esto nos podrá ahorrar tiempo. Una vez que se hayan definido los requisitos por parte del usuario y se ha pasado de fase no se podrá cambiar o aumentar los requisitos, a menos que regresemos a la fase de requisitos y sigamos la cascada.

Las fases son las siguientes:

- Análisis de requisitos.
- Diseño del Sistema.
- Codificación.
- Pruebas.
- Implementación.
- Mantenimiento.

ANÁLISIS

La fase de análisis define los requisitos del software que hay que desarrollar. Inicialmente, esta etapa comienza con una entrevista al cliente, que establecerá lo

que quiere o lo que cree que necesita, lo cual nos dará una buena idea global de lo que necesita, pero no necesariamente del todo acertada. Aunque el cliente crea que sabe lo que el software tiene que hacer, es necesaria una buena habilidad y experiencia para reconocer requisitos incompletos, ambiguos, contradictorios o incluso necesarios. Es importante que en esta etapa del proceso de desarrollo se mantenga una comunicación bilateral, aunque es frecuente encontrarse con que el cliente pretenda que dicha comunicación sea unilateral, es necesario un contraste y un consenso por ambas partes para llegar a definir los requisitos verdaderos del software. Para ello se crea un informe ERS (Especificación de Requisitos del Sistema).

DISEÑO

En esta etapa se pretende determinar el funcionamiento de una forma global y general, sin entrar en detalles. Uno de los objetivos principales es establecer las consideraciones de los recursos del sistema, tanto físicos como lógicos. Se define por tanto el entorno que requerirá el sistema, aunque también se puede establecer en sentido contrario, es decir, diseñar el sistema en función de los recursos de los que se dispone.

En la fase de diseño se crearán los diagramas de casos de uso y de secuencia para definir la funcionalidad del sistema. Con todos esos diagramas e información se obtendrá el cuaderno de carga.

CODIFICACIÓN

La fase más obvia en el proceso de desarrollo de software es sin duda la codificación. Es más que evidente que una vez definido el software que hay que

crear haya que programarlo. Gracias a las etapas anteriores, el programador contará con un análisis completo del sistema que hay que codificar y con una especificación de la estructura básica que se necesitará, por lo que en un principio solo habría que traducir el cuaderno de carga en el lenguaje deseado para culminar la etapa de codificación, pero esto no es siempre así, las dificultades son recurrentes mientras se modifica. Por supuesto que cuanto más exhaustivo haya sido el análisis y el diseño, la tarea será más sencilla, pero nunca está exento de necesitar un re análisis o un rediseño al encontrar un problema al programar el software.

PRUEBAS

Con una doble funcionalidad, las pruebas buscan confirmar que la codificación ha sido exitosa y el software no contiene errores, a la vez que se comprueba que el software hace lo que debe hacer, que no necesariamente es lo mismo. No es un proceso estático, y es usual realizar pruebas después de otras etapas, como la documentación. Generalmente, las pruebas realizadas posteriormente a la documentación se realizan por personal inexperto en el ámbito de las pruebas de software, con el objetivo de corroborar que la documentación sea de calidad y satisfactoria para el buen uso de la aplicación. En general, las pruebas las realiza, idílicamente, personal diferente al que codificó la aplicación, con una amplia experiencia en programación, personas capaces de saber en qué condiciones un software puede fallar de antemano sin un análisis previo.

IMPLEMENTACION

Una vez que tenemos nuestro software, hay que prepararlo para su distribución. Para ello se implementa el software en el sistema elegido o se prepara para que se

implemente por sí solo de manera automática. Cabe destacar que en caso de que nuestro software sea una versión sustitutiva de un software anterior es recomendable valorar la convivencia de sendas aplicaciones durante un proceso de adaptación.

MANTENIMIENTO

Son muy escasas las ocasiones en las que un software no vaya a necesitar de un mantenimiento continuado. En esta fase del desarrollo de un software se arreglan los fallos o errores que suceden cuando el programa ya ha sido implementado en un sistema y se realizan las ampliaciones necesitadas o requeridas. Cuando el mantenimiento que hay que realizar consiste en una ampliación, el modelo en cascada suele volverse cíclico, por lo que, dependiendo de la naturaleza de la ampliación, puede que sea necesario analizar los requisitos, diseñar la ampliación, codificar la ampliación, probarla, documentarla, implementarla y, por supuesto, dar soporte de mantenimiento sobre la misma, por lo que al final este modelo es recursivo y cíclico para cada aplicación y no es un camino rígido de principio a fin. (Casado Iglesias, 2014)

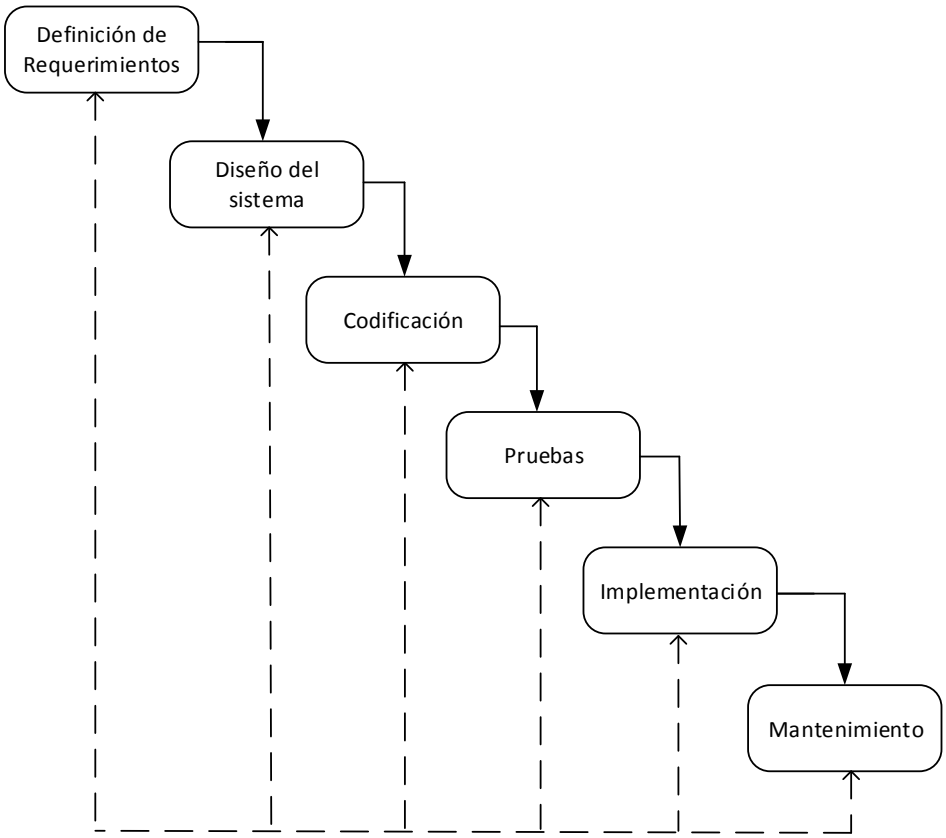


*Ilustración 0.10 Cascada (Tobar J.; 2016)*

## 2.2 Herramientas

### 2.2.1 Lenguaje de software

#### *2.2.1.1 Generaciones de los lenguajes de programación*

Durante la evolución de los lenguajes de programación se han creado 5 generaciones, en cada generación la sintaxis ya se ha simplificado para hacerla más entendible y más parecida al lenguaje natural.

**Primera generación: Código máquina**

Los lenguajes de primera generación, también llamados lenguajes máquina, se caracterizan por utilizar ceros (0) y unos (1) para describir los programas. Estos, en sus inicios, eran introducidos en el ordenador mediante interruptores del tablero frontal del aparato. Ejemplo de línea de código máquina: 10110000 01100001.

La ventaja principal de este tipo de lenguajes es que utilizan instrucciones que pueden ser ejecutadas directamente por el procesador y, por lo tanto, son muy rápidos. Por otra parte, son mucho más complejos de aprender, utilizar y, sobre todo, depurar para buscar errores. Aparte, también encontramos que la portabilidad de estos lenguajes es prácticamente nula: para reutilizar un programa de un ordenador en otro hay que rescribirlo completamente ya que es posible que los procesadores acepten lenguajes máquina diferentes o incluso que tengan arquitecturas diferentes. La gran mayoría de los lenguajes que fueron surgiendo posteriormente, al final y de forma transparente para el programador, acaban traduciendo de una manera u otra el código a código máquina. Eso se hace para superar las dificultades enunciadas anteriormente. "Hello World" es un programa que presenta por pantalla el mismo mensaje.

**Segunda generación: Lenguaje ensamblador**

Para evitar tener que utilizar el alfabeto de la máquina (ceros "0" y unos "1"), que está muy lejos de lo que entendemos nosotros, se crearon los llamados lenguajes ensamblador. Éstos cartografían directamente cada instrucción en el

conjunto de ceros y unos correspondientes. Eso se hace mediante una herramienta llamada ensamblador. La arquitectura de un ordenador se refiere normalmente a la estructura interna del procesador. Los lenguajes ensamblador son totalmente dependientes de la familia del procesador y de su arquitectura, igual que el código máquina (por ejemplo, un programa hecho para Intel no sirve para AMD o Motorola, ya que son diferentes fabricantes de procesadores). Ejemplo de línea de código ensamblador: mov al, 061h.

Aunque claramente la introducción de los lenguajes ensamblador implicaba una mejora importante en la legibilidad del programa, apareció una nueva necesidad entre los programadores: poder programar pensando más en cómo solucionar el problema desde el punto de vista humano que en el funcionamiento interno del ordenador. De ahí surgen los conceptos de lenguajes de programación de bajo y de alto nivel. Cuanto más de alto nivel se considera un lenguaje, más cerca se sitúa del lenguaje natural y de la manera humana de razonar. Las principales aplicaciones de estos lenguajes son los kernel's[5], los drivers[6] de los dispositivos y las librerías de sistemas. El "Hello World" para lenguaje ensamblador tendría el código que se presenta a continuación. Este ejemplo está escrito para ensamblador de un procesador Intel x86 con DOS y utilizando TASM.

```
MODEL SMALL
IDEAL
STACK 100H
```

[5] Núcleo de un sistema operativo.
[6] Programas que permiten al sistema operativo comunicarse con los periféricos.

```
DATASEG
MSG DB ‘Hello, world!’, 13, ‘$’
CODESEG
Start:
MOV AX, @data
MOV DS, AX
MOV DX, OFFSET MSG
MOV AH, 09H ; DOS: output ASCII$ string
INT 21H
MOV AX, 4C00H
INT 21H
END Start
```

Es normal que, a simple vista, no se entienda, ya que estamos trabajando en las partes y los mecanismos internos del procesador de un ordenador. Por eso se llaman lenguajes de bajo nivel, porque se baja al nivel de la máquina.

**Tercera generación: Lenguajes de alto nivel**

Viendo la complejidad de programar con los lenguajes que había hasta entonces, empezaron a aparecer los lenguajes de alto nivel. Estos aportan una legibilidad mucho más próxima al lenguaje natural y ahorran al programador el tener que pensar en cada orden mínima que tiene que ejecutar el ordenador. Cualquier instrucción de este tipo de lenguajes equivale a unas cuantas líneas de código máquina o ensamblador. Ejemplo de línea de código C: if (x>0) factorial(x).

Así, a partir del código escrito en un lenguaje de este grupo, se utiliza una herramienta que "traduce" el código a código máquina. Normalmente se hace mediante un compilador, aunque puede utilizarse un intérprete o incluso una máquina virtual. La mayor parte de los lenguajes que se utilizan hoy día para la programación de aplicaciones son de tercera generación (por ej. C, C++ y Java).

**Cuarta generación (4GL): Lenguajes casi naturales**

Son lenguajes de programación que no son de propósito general, como los que había habido hasta ahora, sino que están diseñados para algún propósito específico (por ej. Sculptor, Dbase, SAS, PHP, etc.). Los lenguajes de tercera generación habían conseguido facilitar mucho la comunicación entre el hombre y la máquina, y habían mejorado claramente el desarrollo del software desde sus predecesores. Aun así, la tarea de programar se seguía considerando frustrante, lenta y propensa a errores. Lo cual llevó a la primera gran crisis del software, ya que la cantidad de trabajo que se esperaba que hicieran los programadores superaba con creces el tiempo que tenían para llevarla a cabo. Para solucionar el problema, se tomaron varias medidas, entre éstas encontramos la definición de la ingeniería del software y el diseño de lenguajes de programación según los tipos de problemas que debían resolverse. Ejemplo de línea de código de SQL:

SELECT * FROM USUARIO WHERE NOMBRE = "BOB".

Estos lenguajes están diseñados para reducir el esfuerzo de programación, el tiempo y el coste de desarrollo. Pero no se puede errar en la elección del lenguaje, ya que si no el proyecto se convierte muy probablemente en un

fracaso. No hay lenguajes mejores ni peores. Lo que puede facilitar o complicar la programación de una aplicación es la idoneidad del lenguaje elegido para codificar.

Algunos tipos de lenguajes de cuarta generación son:

- Tratamiento de bases de datos (por ej. Progress4gl, SQL y FOCUS)
- Generadores de informes (por ej. Progress4gl, PostScript, BuildProfessional y Gauss)
- Manipuladores y analizadores de datos (por ej. Progress4gl, Informix-4GL, Mathematica y SAS)
- Tratamiento de flujo de datos (por ej. APE y AVS)
- Generación y dibujo de pantallas (por ej. Oracle Forms y Unify Accell)
- Generadores de interfaces gráficas de usuario (por ej. Progress4gl, Borland Delphi, Visual Basic's form editor, Windows Forms y OpenROAD)
- Desarrollo web (por ej. PHP y ASP)

**Quinta generación (5GL): Inteligencia artificial**

Diferentes fuentes definen la quinta generación de lenguajes de diversas maneras. Entre éstas, las dos principales son: Lenguajes para inteligencia artificial, son aquellos basados en la resolución de problemas que utilizan restricciones para modelarlos, sin programar un algoritmo que lo haga. La mayoría son lenguajes lógicos, basados en restricciones, y/o declarativos (por ej. Prolog, ILOG o eCLiPse). El programador no se tiene que preocupar por CÓMO resolver sino de QUÉ resolver. Se hicieron bastantes inversiones en estos lenguajes a lo largo de los años noventa, con la idea de sustituir los lenguajes de alto nivel, pero fue un fracaso y actualmente se utilizan sólo en

círculos académicos. Ejemplo de línea de código: hermano(X,Y):-madreOpadre(X,Z), madreOpadre(Y,Z).

Esta línea dice que X e Y son hermanos si comparten padre o madre.

En el ejemplo anterior se utiliza una regla. Los lenguajes lógicos utilizan hechos y reglas para modelar la realidad y a partir de ésta, poder inferir las respuestas a posibles preguntas. Otros presentan los 5GL como los que utilizan una interfaz gráfica o visual de usuario para crear un código fuente que se compila posteriormente con un compilador de lenguajes de tercera o cuarta generación. Compañías como Borland, Microsoft e IBM elaboran este tipo de productos de programación visual para el desarrollo de aplicaciones en Java, por ejemplo.

(Noguera Otero & Riera Terrén, 2013)

### 2.2.2 Detalle de lenguaje seleccionado

El sistema debe estar incluido en el ERP MFG/PRO eB2 que es utilizado por la empresa, por lo que el desarrollo debe ser realizado en el lenguaje Progress4GL.

#### *2.2.2.1 PROGRESS 4gl o OpenEdge Advanced Business Language*

Progress es un lenguaje de desarrollo de aplicaciones de negocio, creada por Progress Software Corporation (PSC). Este lenguaje utiliza palabras de Ingles como sintaxis, para simplificar el desarrollo de software.

El lenguaje fue llamado PROGRESS o PORGRESS4GL hasta la versión 9, pero en el 2006, PSC cambio el nombre a OpenEdge ABL (OpenEdge

Advanced Business Language), con el fin de superar la presunción que el lenguaje 4GL es menos capaz que otros lenguajes.

ABL (Advanced Business Language) es un lenguaje de programación procedimental y orientado a objetos de alto nivel que permite construir casi todos los aspectos de una aplicación de negocios de la empresa, desde la interfaz de usuario hasta el acceso a bases de datos y la lógica de negocio. ABL es una herramienta versátil y extraordinariamente poderosa. No sólo se puede utilizar para desarrollar aplicaciones, se pueden construir muchas de las herramientas que se utilizan para ayudar a crear y apoyar a estas aplicaciones.

ABL incluye sentencias y palabras clave de gran alcance que ayudan a conseguir su objetivo, el desarrollo de aplicaciones empresariales. Las sentencias o palabras clave en ABL pueden hacer el trabajo de docenas o posiblemente cientos de líneas de código en un lenguaje de 3ª Generación. Por ejemplo, en una sola instrucción ABL se pueden acceder a los datos existentes en una base de datos, así como aplicar cambios a los datos en la base de datos. Otras palabras claves permiten programar con gran precisión. Esta flexibilidad es lo que da a ABL su gran poder como un lenguaje de programación. La mayoría de las herramientas de desarrollo que se utilizan para desarrollar aplicaciones OpenEdge son ellas mismos escritos en ABL.

El ABL deriva su poder en la capacidad de expresar de forma concisa los procesos de negocio. Estas operaciones de negocios o entidades suelen tener un componente significativo de procesamiento de datos. OpenEdge incluye herramientas de programación, herramientas de bases de datos y recursos de

servidor en un entorno que le permite tener fácil acceso a todos los recursos que componen su aplicación.

OpenEdge ABL ayuda a los desarrolladores a crear aplicaciones usando su propia base de datos relacional integrada. Estas aplicaciones son portables y permiten acceder a varias bases de datos sin tener que aprender los métodos de acceso.

OpenEdge ABL es un lenguaje de programación procedimental. Eso significa que se escribe conjuntos de instrucciones del lenguaje que se pueden guardar en los procedimientos individuales. Las sentencias se ejecutan normalmente o procesan en el orden en que aparecen en el procedimiento. En un procedimiento sencillo, las sentencias se ejecutan tal y como aparecen.

Un procedimiento de OpenEdge ABL se compone de bloques. El procedimiento en sí es el bloque principal del procedimiento. Existen varias maneras de definir otros bloques dentro del bloque principal. La sentencia FOR EACH con su sentencia de cierre END es un ejemplo de un bloque anidado, en este caso la sentencia itera un conjunto de registros de base de datos y ejecuta todo el código en el medio de cada registro en el conjunto. Hay otras sentencias de bloques que se pueden utilizar para diferentes propósitos.

Un procedimiento de OpenEdge ABL se compone de una secuencia de instrucciones del lenguaje. Cada sentencia tiene una o más palabras clave, junto con otros símbolos tales como nombres de campo de base de datos o nombres de variables. Una sentencia normalmente termina con un punto. Por convención, una sentencia que define el comienzo de un bloque, como por ejemplo, la sentencia FOR EACH, puede terminar con dos puntos. No hay

ninguna importancia para el final de la línea en Progress. Una sola sentencia puede ocupar varias líneas, y si lo desea, puede haber varias sentencias en una sola línea. No hay ninguna sintaxis especial que se necesita para romper una sentencia entre líneas.

Debe haber al menos un espacio (u otro carácter de espacio como una tabulación o una nueva línea) entre cada token7 en la sentencia. OpenEdge no es sensible a los espacios en blanco adicionales. Es decir, se puede agregar varios espacios entre líneas de código y poner una tabulación entre las partes de una declaración para facilitar la lectura, y nada de esto va a afectar a cómo se ejecuta la instrucción.

Al definir procedimientos, variables, tablas de bases de datos y los campos, se debe asignar nombres que comienzan con una letra, después de eso, el nombre puede incluir letras, dígitos numéricos, puntos, guiones (-) y guiones bajos (_). Tenga en cuenta que, si bien el propio lenguaje tiene muchas palabras clave separadas por guiones, no se debe utilizar guiones en los nombres de las tablas de base de datos y en los campos, así como el procedimiento y los nombres de las variables en la aplicación.

Una buena convención de nomenclatura para todo tipo de nombres es el uso de mayúsculas y minúsculas, como por ejemplo: el campo CliCodigo, para romper un nombre en varias palabras u otras partes identificables, utilizando las letras

[7] Un **token** o también llamado componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación. Ejemplo: palabras clave (if, else, while, int, ...), identificadores, números, signos, o un operador de varios caracteres, (por ejemplo, :=).

mayúsculas para identificar el comienzo de cada palabra o la subparte en el nombre para mejorar la legibilidad del código.

Hay tres tipos básicos de sentencias en un programa: sentencias de procedimiento, las sentencias de acceso a base de datos, y las sentencias de interfaz de usuario. A veces las sentencias individuales contienen elementos de los tres tipos e ilustra el poder del lenguaje.

La sentencia FOR EACH es un acceso a la base de datos, sin embargo, este define un resultado en cada iteración. Esta simple convención de combinar la lógica de procedimiento con acceso a la base de datos es una característica fundamental y enormemente poderosa. En otro lenguaje, se tendría que definir una consulta de base de datos utilizando una sintaxis que básicamente separa el lenguaje del procedimiento con su alrededor, a continuación tienen declaraciones adicionales para controlar el flujo de datos en el procedimiento. En el 4GL, puede combinar todos estos elementos en una forma que es muy natural, flexible y eficiente, y se relaciona con la manera de pensar acerca de cómo el programa utiliza los datos.

La sentencia DISPLAY muestra que las sentencias de interface de usuario están integradas con el resto del programa. Progress contiene sentencias no solo para visualizar la información de la base de datos, sino también puede crear, actualizar y eliminar registros. (Progress OpenEdge, 2016)

# Capítulo 3: Modelo de empresas de consumo masivo

## 3.1 Giro del negocio

La empresa de consumo masivo es una empresa comercial e industrial, que se encuentra dentro de las industrias de alimentos, actualmente produce en el Ecuador varios productos como refrescos congelados, shampoo para hombres y mujeres, gel para cabello e importa y comercializa productos como jugos en polvo, pasta dental, bebidas, para cuidados del hogar desde Colombia en donde se encuentra la matriz de la multinacional.

Por la variedad en el portafolio de productos se han definido diferentes canales de venta para cubrir y dominar todo el mercado en el país, por lo que cada canal de venta tiene sus particularidades como por ejemplo: el número de clientes, capacidad de pago, términos de crédito, días de caducidad del producto, etc.

## 3.2 Tipo de operación de la empresa

En Ecuador la empresa lleva más de 10 años desde sus inicios. La empresa tiene varios puntos de venta en todo el país, sin embargo se tiene únicamente dos centros de distribución para todo el país, ya que almacenar el producto en cada punto de venta representa una gran pérdida de tiempo y/o dinero por los costos del almacenaje y la caducidad del mismo. El centro de distribución de Quito se encarga de la distribución al norte del país y el de Guayaquil de la distribución al sur del país.

Para cubrir la venta en todo el mercado la empresa se ha dado cuenta que el trato a cada cliente va a depender de las características del mismo, por lo que se han definido varios canales de ventas, los cuales son:

- Preventa. Es el canal que se encarga de la venta y distribución a las tiendas de barrio y establecimientos de consumo en sitio.
- Mayoristas. Es canal que se encarga de la venta y distribución a grandes tiendas en donde almacenan el producto para ser vendido a pequeños comerciantes.
- Clientes especiales. Es el canal que se encarga de la venta y distribución a las grandes cadenas de consumo masivo.
- Microaliados. Es el canal que se encarga de la distribución a grandes tiendas que se encuentran fuera de las grandes ciudades como Quito, Guayaquil, Ambato, Cuenca.

Cada canal de venta tiene sus propias negociaciones o características, como por ejemplo: vendedores, precios, días de entrega, días de caducidad, etc.

A lo largo de la historia de la empresa el almacenaje del inventario ha ido cambiando según las necesidades y la realidad del país, en la actualidad se ha definido que el almacenaje se lo va a realizar mediante un operador logístico, ya que eso permite enfocarse en el giro del negocio y no desgastarse en un servicio que puede ofrecer un proveedor.

Para no perder el total control del inventario de la empresa se definió que se debe adicionar un almacén con inventario virtual que será el centro de distribución, en donde el inventario del almacén virtual debe ser similar al inventario físico del OPL, se mantendrán los almacenes de facturación, sin embargo el uso de estos almacenes se limitará al proceso de facturación para no afectar el inventario del almacén virtual,

todo movimiento de inventario que se realice lo deben realizar las dos empresas para no tener diferencias de inventario.

## 3.3 Problemática actual

En la actualidad el intercambio de información con el operador logístico se lo realiza por medio de correo electrónico mediante archivos Excel que son generados manualmente, dando cabida a errores involuntarios y pérdida de tiempo.

Al momento de verificar el inventario para la facturación los facturadores obtienen un reporte con todo el producto que se va a facturar, se genera un archivo en Excel con el resultado del reporte y se verifica manualmente que los productos y sus cantidades sean las correctas. Una vez obtenido el archivo se envía al OPL y se debe esperar que los superiores del OPL realicen una verificación manual del producto que se solicitó cumpliendo los días de caducidad, este proceso tarda alrededor de unas 2 horas ya que se debe ir a revisar en la bodega, con el resultado de la verificación manual se genera un archivo en Excel con la respuesta del producto solicitado y se envía a la empresa para que pueda continuar con su proceso de facturación, en el caso de no tener el inventario necesario los facturadores de la empresa deben disminuir las cantidades a embarcar en todos los pedidos[8].

Una de las principales problemáticas al momento de verificar el producto es que los inventarios no son similares, por lo que se tiene diferencias al momento de verificar,

[8] Embarcar pedidos es el proceso en el cual se disminuye el inventario de un producto y se lo asigna en su respectivo pedido.

esto causa varios reprocesos al momento verificar manualmente el producto y al momento de embarcar pedidos.

En ocasiones se presenta algunos accidentes al momento que el OPL realiza el Picking[9], como por ejemplo que de derribe una caja y se destruya el producto, por el cual no se puede alistar la cantidad del producto que ya fue verificada y existen diferencias entre el producto alistado por el OPL y el producto facturado por la empresa.

## 3.4 Mejoras propuestas

Diseñar y desarrollar un sistema que permita al OPL verificar automáticamente el inventario que se requiere para facturar los pedidos de los distintos canales de venta, disminuyendo/reemplazando el número de pasos a seguir para facturar los todos los pedidos del día y de esta manera optimizar los recursos horas-hombre y disminuir el tiempo de ejecución de 2 horas a 30 minutos aproximadamente por todos los consolidados, adicionalmente se desarrollará reportes que comparen los inventarios de la empresa vs del OPL, esto permitirá tener un inventario similar todos los días disminuyendo la perdida de producto y el reproceso de la verificación física del producto.

Para la verificación del inventario a facturar se desarrollará una opción que permita generar consolidados de productos y enviar el archivo con la información necesaria

[9] Picking es un proceso básico en el alistamiento de productos para cada pedido, este es un proceso físico realizado en los almacenes.

para que el OPL pueda realizar el proceso de verificación automáticamente en su sistema, una vez el OPL verifique el inventario y genere el archivo de respuesta, el sistema obtendrá la información de ese archivo y guardará las cantidades verificadas.

Se desarrollará una opción que permita realizar una transferencia por consolidados de las cantidades verificadas por el OPL del almacén virtual al almacén de facturación, se transferirá únicamente el producto verificado ya que es el verdadero producto con el que se cuenta para la facturación.

Se desarrollará una opción que permita embarcar las cantidades verificadas por el OPL, agotando los faltantes en caso de existir y de esta manera disminuir el reproceso de hacer el embarque manual en los pedidos con faltantes.

Se desarrollará una opción que obtenga automáticamente de un archivo suministrado por el OPL la información del inventario de la empresa, con dicha información se realizará una comparación del inventario virtual vs el inventario que tiene el OPL, en caso de existir una diferencia se marcará el producto con la diferencia.

Se desarrollará una opción que obtenga automáticamente de un archivo suministrado por el OPL la información del producto disminuido por el OPL una vez que se haya realizado el Picking del día, con dicha información se realizará una comparación del producto facturado vs el producto disminuido por parte del OPL, en caso de existir diferencias se marcará el producto con la diferencia.

# Capítulo 4: Diseño y desarrollo de la interface

## 4.1 Análisis de requisitos

### 4.1.1 Introducción

En el presente documento se explicarán y analizarán los requerimientos para el Proyecto “Interface en doble vía de una empresa de venta de consumo masivo/operador logístico”.

#### *4.1.1.1 Propósito*

El documento tiene como propósito dar a conocer el funcionamiento general de la interface en doble vía de una empresa de venta de consumo masivo y el operador logístico, dicho documento está dirigido a los desarrolladores de las empresas involucradas y usuarios finales.

#### *4.1.1.2 Ámbito del Sistema*

El nombre del sistema: AVIF “Automatización de Verificación de Inventarios a Facturar”.

El sistema de encargará del proceso de verificación del inventario para la facturación de pedidos de todos los canales de ventas, cumpliendo la política de despacho definida para cada canal, comparación de inventarios, reportes de consolidados, seguimiento al producto facturado.

El principal beneficiado con el sistema es el área de logística.

El objetivo es automatizar el proceso de verificación de inventario para disminuir y optimizar tiempos en el proceso de facturación, adicionalmente se realizará comparaciones de inventarios.

#### *4.1.1.3 Definiciones, Acrónimos y Abreviaturas*

- OPL (Operador Logístico). son las empresas que se encargan de almacenaje, alistamiento, distribución, venta de producto, etc.
- Canal de venta. Agrupación de clientes con las semejantes características y con un trato distinto.
- PT. Producto terminado
- Prevta. Clientes de tiendas.
- Embarque de pedidos. Asignar producto del inventario a cada pedido.

#### *4.1.1.4 Referencias*

Para realizar este documento se ha tomado en cuenta la experiencia de los usuarios expertos del área de logística y distribución.

#### *4.1.1.5 Visión General del Documento*

El documento está dividido en 4 secciones:

– La sección 1 se enfoca en la explicación, objetivos, metas y descripción del documento.

- La sección 2 está orientada, como su nombre lo indica, a la descripción general del sistema, donde la información está orientada al cliente/usuario final.
- La sección 3 trata sobre los requisitos específicos. Se emplean términos técnicos orientados principalmente a los desarrolladores y programadores.
- La sección 4 son los apéndices, contiene el flujo del proceso de facturación.

### 4.1.2 Descripción General

#### *4.1.2.1 Perspectiva del Producto*

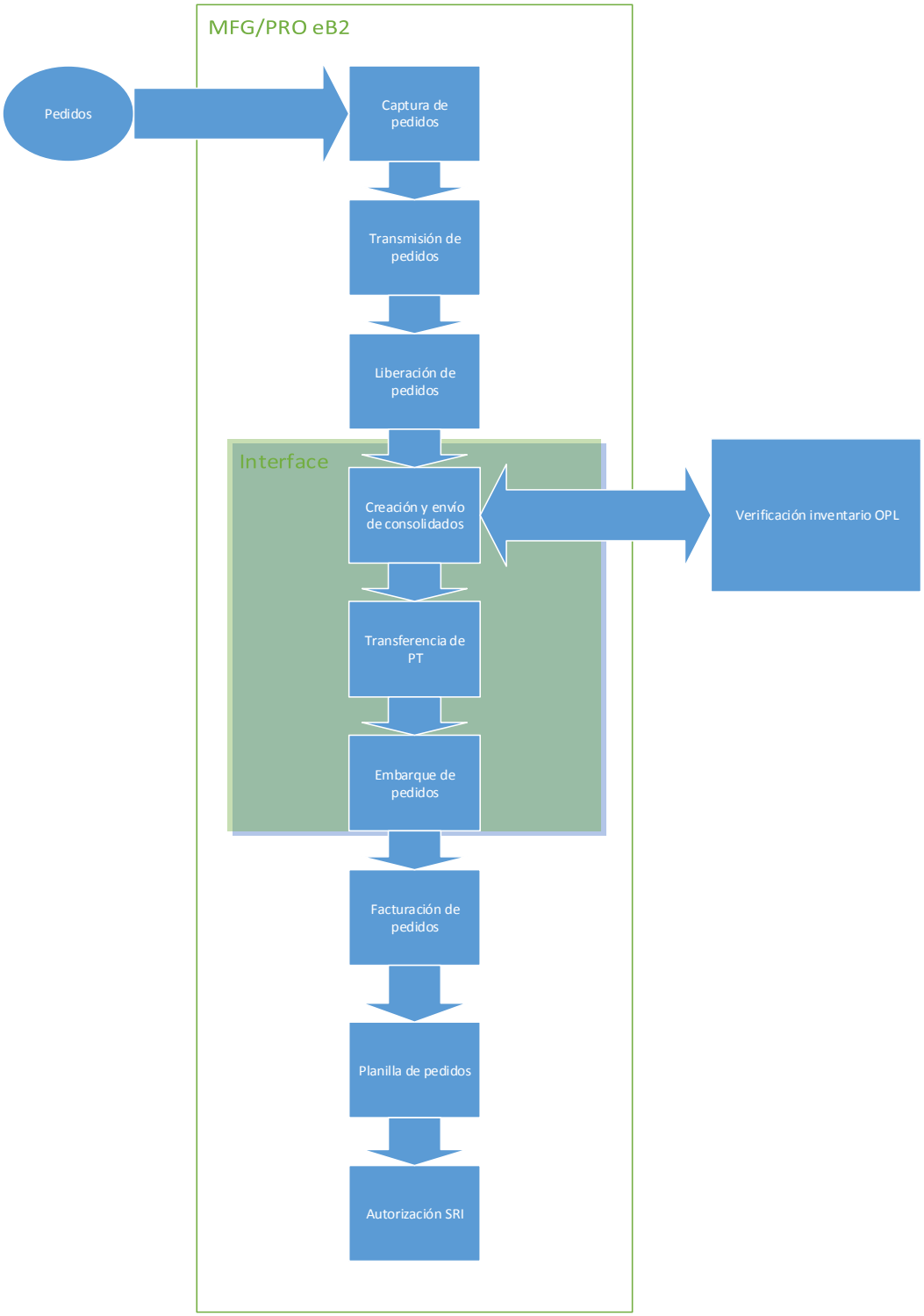


*Ilustración 0.1 Perspectiva Producto (Tobar J.; 2016)*

#### *4.1.2.2 Funciones del Producto*

- Parametrización del sistema.
- Generación y envío de consolidados
- Reporte Inventarios
- Reportes consolidados
- Transferencia de Consolidados
- Embarque de consolidados

#### *4.1.2.3 Características de los Usuarios*

El sistema cuenta con tres tipos de usuarios:

- El primer tipo de usuario es el Usuario Líder, son personas con un nivel de estudio superior, quien se encarga de la correcta parametrización y ejecución del sistema.
- El segundo tipo de usuario son los Facturadores, son personas con un nivel de estudio secundario, quienes se encargan de ejecutar una gran cantidad de actividades para asegurar el proceso de facturación.
- El tercer tipo de usuario es el Supervisor, son personas con un nivel de estudio superior, quien es el encargado del inventario, es decir, debe verificar que siempre este cuadrado el inventario de la empresa de consumo masivo y el OPL, verificar los movimientos de inventarios, etc.

#### *4.1.2.4 Restricciones*

- El formato de los archivos de intercambio es .csv.

- La transferencia de archivos se la debe realizar de manera automática y/o manual.
- El acceso a las opciones se debe limitar según el cargo de cada tipo de usuario.
- El sistema debe ser multiusuario, ya que existen dos centros de distribución que pueden trabajar al mismo tiempo.
- Para el intercambio de archivos se debe utilizar las herramientas ya definidas y que ya han sido desarrolladas por la empresa, es decir el intercambio de archivos se lo realiza por FTP.
- Cada proceso realizado de los consolidados debe tener un estado diferente para realizar la trazabilidad del mismo.

#### *4.1.2.5 Suposiciones y dependencias*

- La empresa debe contar con un proceso automático de intercambio de archivos.
- El inventario del almacén virtual del OPL debe ser igual al inventario físico del OPL.
- Los almacenes de facturación no deben tener inventario ya que son almacenes para facturar y no para almacenaje.

#### *4.1.2.6 Requerimientos futuros*

Una de las necesidades para complementar el proyecto es que todos los movimientos de inventarios cuenten con una interface para tener un proceso completamente automático, es decir al momento que se realice una nota de crédito, transferencia de inventarios, importación de producto se genere un archivo que le permita al OPL ingresar automáticamente esos movimientos en su sistema para tener un inventario en línea.

### 4.1.3 Requerimientos Específicos

#### *4.1.3.1 Interfaces Externas*

La interfaz gráfica con la que el usuario final interactúa deberá ser intuitiva de manera que, sin un manual de uso, el usuario identifique rápidamente los componentes y las secciones del sistema.

#### *4.1.3.2 Requisitos Funcionales*

##### 4.1.3.2.1 Parametrización consolidados

- Introducción: El sistema debe estar en la capacidad de cumplir las necesidades del negocio según su crecimiento, por lo que se requiere que los parámetros se los pueda modificar según la necesidad del negocio.
- Entrada:
  - Almacén y ubicación de facturación.
  - Característica.
  - Descripción.
  - Estado.
  - Almacén y ubicación virtual del OPL.
  - Nombre del almacén y ubicación del OPL.
  - Fecha de inicio.
  - Fecha de fin.
- Proceso: Crear una relación entre los almacenes y ubicaciones de facturación, virtual del OPL y el físico del OPL.

- Salida: N/A.

#### 4.1.3.2.2 Generación y envío de Consolidados

- Introducción: En la actualidad se envían consolidados por días de caducidad.
- Entrada:
  - Almacén y ubicación de facturación.
  - Búsqueda y selección de pedidos.
  - Listado de los pedidos que están en el consolidado.
- Proceso: Buscar y seleccionar los pedidos que se deseen facturar para crear un consolidado de pedidos y PT, cada PT debe tener la cantidad solicitada y los días de caducidad.
- Salida: Devolver el número de consolidado y enviar automáticamente el archivo con las cantidades solicitadas por cada PT.

#### 4.1.3.2.3 Transferencia de Consolidados

- Introducción: Las transferencias de producto entre almacenes se las debe realizar masivamente con un identificativo de transferencia para el control de inventario.
- Entrada:
  - Almacén y ubicación origen
  - Almacén y ubicación destino
  - El número de consolidado

- Proceso: Transferir las cantidades confirmadas por el OPL del almacén origen hacia el almacén destino, se debe validar que los almacenes y ubicaciones ingresadas sean las que se parametrizaron.
- Salida: Presentar un mensaje de transferencia exitosa, caso contrario el error presentado.

#### 4.1.3.2.4 Embarque de Consolidados

- Introducción: El embarque es el proceso en donde se le asigna el producto a cada pedido, es decir, se disminuye del inventario el producto a facturar, para posteriormente ser facturado.
- Entrada: Número de consolidado.
- Proceso: Embarcar todos los productos con las cantidades verificadas por el OPL a sus respectivos pedidos que se encuentra dentro del consolidado.
- Salida: Presentar un mensaje de embarque exitoso, caso contrario el error presentado.

#### 4.1.3.2.5 Reporte Consolidados

- Introducción: Se necesita conocer el estado de los consolidados que se generan y su detalle para continuar el proceso.
- Entrada:
  - Almacén
  - Fecha
  - Consolidado
  - Canal

- Estado de los consolidados
- Detalle de reporte

- Proceso: Consultar el detalle del consolidado según la selección del detalle del reporte, se va a tener 3 opciones para seleccionar, un detalle de pedidos, detalle de producto verificado y el detalle de los consolidados. El reporte debe tener una cabecera con la información del consolidado seguido del detalle que uno seleccione.
- Salida:
  - Detalle de pedidos:
    - Cabecera:
      - Almacén
      - Número consolidado
      - Estado
    - Detalle:
      - Pedidos

  - Detalle de productos:
    - Cabecera:
      - Almacén
      - Número consolidado
      - Estado
    - Detalle:
      - Código de PT
      - Descripción del PT
      - Cantidad pedida

- Cantidad confirmada

- Detalle de consolidado:
  - Almacén
  - Número consolidado
  - Estado
  - Fecha
  - Días de caducidad.

#### 4.1.3.2.6 Reporte Inventarios

- Introducción: Se necesita contar con un reporte que nos permita realizar la comparación del inventario virtual y el físico, para disminuir la perdida de dinero por diferencias de inventario.
- Entrada:
  - Almacén
  - Ubicación
  - Referencia
- Proceso: Se debe extraer la información del inventario proporcionada por el OPL mediante un archivo, con la cual se realizará la comparación de inventarios para verificar que el inventario sea similar.
- Salida:
  - Almacén
  - Ubicación
  - Código del PT
  - Descripción del PT

- Unidad de medida
- Cantidad virtual
- Cantidad física
- Diferencia de cantidades

#### *4.1.3.3 Requisitos de Rendimiento*

Las características del desempeño que se debe tomar en cuenta para el desarrollo del sistema son las siguientes:

- El volumen de registro por día son 2554.
- El número de transacciones a realizar por día son 3.
- El número de usuarios simultáneamente son 2.

#### *4.1.3.4 Otros Requisitos*

Se debe contar con toda la documentación necesaria para parametrizar el sistema, así como un instructivo que permita a los nuevos usuarios del sistema comprender el funcionamiento del sistema.

#### *4.1.3.5 Especificación de estructura de información*

Para el intercambio de archivos se definió lo siguiente:

Los nombres de los archivos que se generan automáticamente serán los siguientes:

Interface Facturación:

- Archivo enviado por la empresa: QCR00000220140613155434

- Archivo enviado por la empresa: OCR000002201406131554 34

En el ejemplo:

- La Q es el carácter de quien envía el archivo, “Q” cuando envía la empresa y “O” cuando envía el OPL.
- CR000002 es el consecutivo del consolidado, este será generado por ERP.
- 2014613 es la fecha cuando se generó el archivo.
- 155434 es la hora en la que se generó el archivo.

Para el consecutivo de los archivos de reporte será de la siguiente manera:

- Reporte inventario: INVTHMLG20140613
- Reporte inventario facturado: INVTFCTR20140614

Interface Facturación

Archivo generado por el ERP (Requerimiento de cantidades para cada producto)

*Tabla 2 Estructura interface facturación (Tobar J.; 2016)*

| Información | Tipo | Tamaño | Ejemplo |
|---|---|---|---|
| *Cabecera* | | | |
| *Estado del Documento* | *Char* | *8* | *“Nuevo”* |
| *Número del consolidado* | *Char* | *8* | *“CO000021”* |
| *Fecha del consolidado* | *Date* | *Mm/dd/aaaa* | *05/27/2014* |
| *Hora del consolidado* | *Time* | | *15:54:34* |
| *Fecha respuesta* | *Date* | *Mm/dd/aaaa* | |
| *Hora respuesta* | *Time* | | |

| | | | |
|---|---|---|---|
| *Detalle* | | | |
| *Número de ítems* | *Integer* | | *1* |
| *Código producto* | *Char* | *8* | *“79090809”* |
| *Descripción* | *Char* | *50* | *“Ego SH”* |
| *Unidad de medida* | *Char* | *2* | *“un”* |
| *Cantidad solicitada* | *Decimal* | *>>>>>>>>>.9<<<<* | *2311* |
| *Cantidad confirmada* | *Decimal* | *>>>>>>>>>.9<<<<* | *0* |
| *Días de vencimiento* | *Integer* | | *60* |

Archivo enviado por el OPL (Confirmación de cantidades de cada producto)

*Tabla 3 Estructura de respuesta por el OPL (Tobar J.;2016)*

| Información | Tipo | Tamaño | Ejemplo |
|---|---|---|---|
| *Cabecera* | | | |
| *Estado del Documento* | *Char* | *8* | *“Confirma”* |
| *Número del consolidado* | *Char* | *8* | *“CO000021”* |
| *Fecha del consolidado* | *Date* | *Mm/dd/aaaa* | *05/27/2014* |
| *Hora del consolidado* | *Time* | | *15:54:34* |
| *Fecha respuesta* | *Date* | *Mm/dd/aaaa* | *05/27/2014* |
| *Hora respuesta* | *Time* | | *15:58:00* |
| *Detalle* | | | |
| *Número de ítems* | *Integer* | | *1* |
| *Código producto* | *Char* | *8* | *“79090809”* |
| *Descripción* | *Char* | *50* | *“Ego SH”* |
| *Unidad de medida* | *Char* | *2* | *“un”* |

| | | | |
|---|---|---|---|
| *Cantidad solicitada* | *Decimal* | *>>>>>>>>>.9<<<<* | *2311* |
| *Cantidad confirmada* | *Decimal* | *>>>>>>>>>.9<<<<* | *2300* |
| *Días de vencimiento* | *Integer* | | *60* |

Interface Homologación de inventarios

*Tabla 4 Estructura interface inventarios (Tobar J.; 2016)*

| Información | Tipo | Tamaño | Ejemplo |
|---|---|---|---|
| *Estado del Documento* | *Char* | *8* | *"Nuevo"* |
| *Almacén* | *Char* | *8* | *"Guayaqui"* |
| *Ubicación* | *Char* | *8* | *"gye01"* |
| *Articulo* | *Char* | *8* | *"79090809"* |
| *Cantidad* | *Decimal* | *>>>>>>>>>.9<<<<* | *57* |
| *Unidad de medida* | *Char* | *2* | *"Un"* |
| *Fecha de vencimiento* | *Date* | *Mm/dd/aaaa* | *05/30/2014* |

### 4.1.4 Apéndices

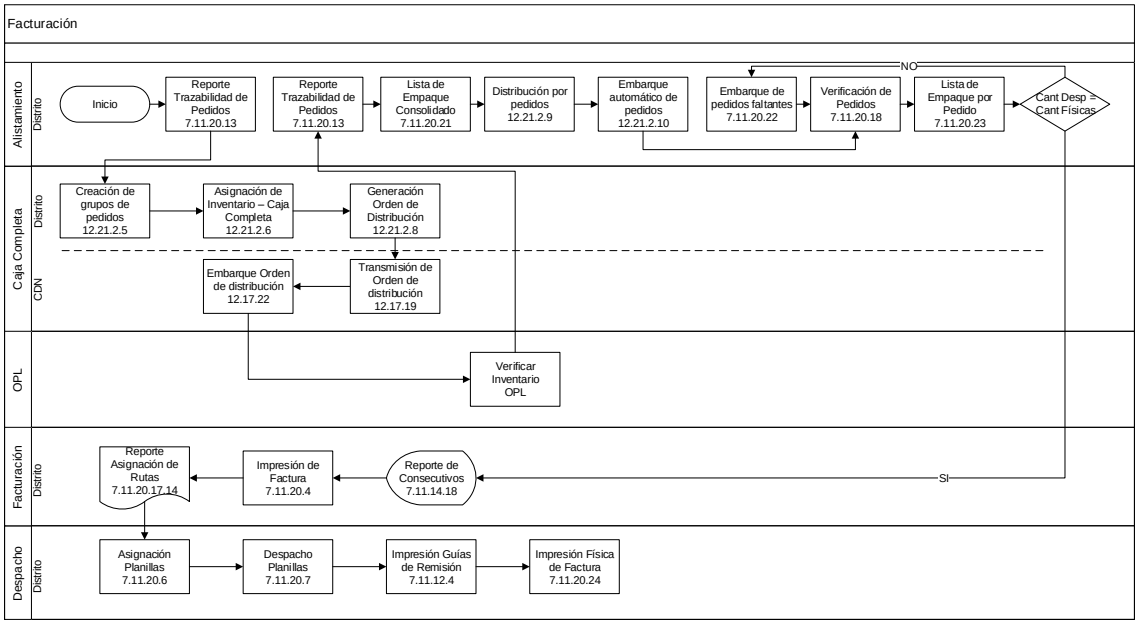


*Ilustración 0.2 Proceso facturación actual (Tobar J.; 2016)*

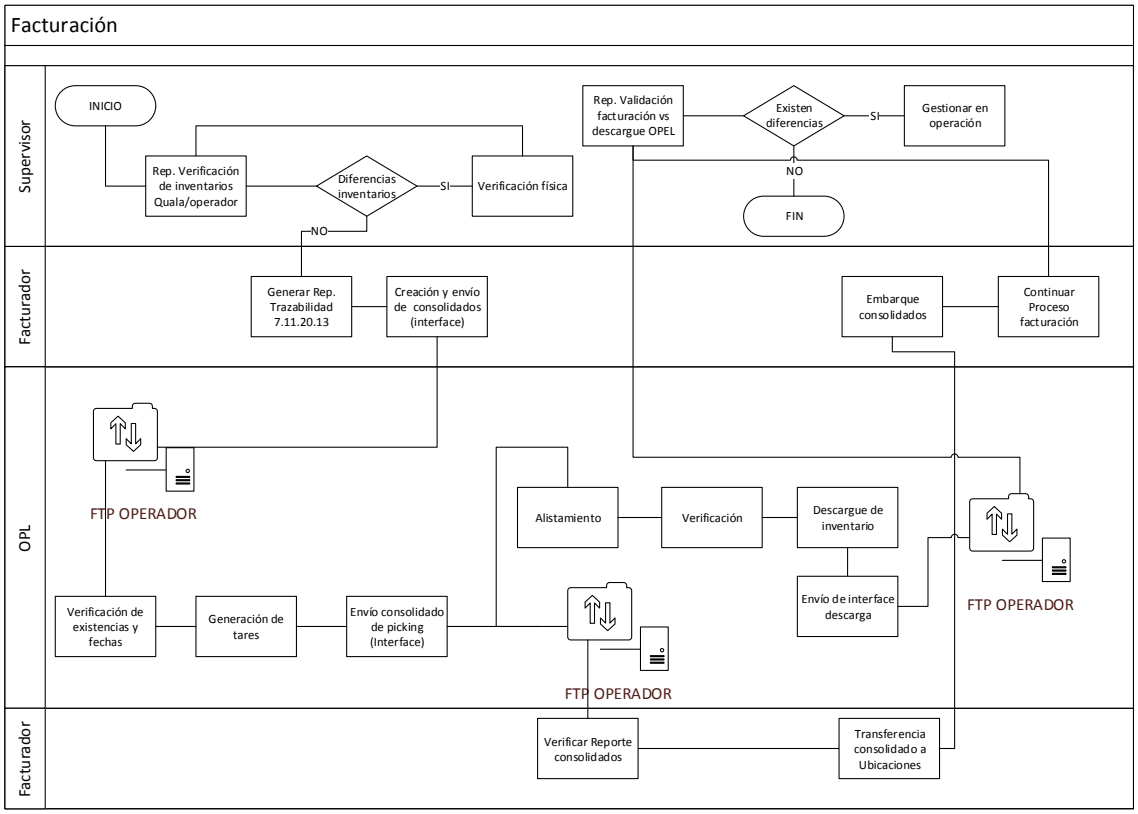


*Ilustración 0.3 Proceso facturación propuesto (Tobar J.; 2016)*

## 4.2 Diseño del Sistema

### 4.2.1 Introducción

En el presente documento se explicaran y analizaran los requerimientos para el Proyecto "Interface en doble vía de una empresa de venta de consumo masivo / operador logístico".

#### *4.2.1.1 Propósito*

El documento tiene como propósito dar a conocer el funcionamiento general de la interface en doble vía de una empresa de venta de consumo masivo y el operador logístico, dicho documento está dirigido a los desarrolladores de las empresas involucradas y usuarios finales.

#### *4.2.1.2 Definiciones, Acrónimos y Abreviaturas*

- OPL (Operador Logístico). son las empresas que se encargan de almacenaje, alistamiento, distribución, venta de producto, etc.
- Canal de venta. Agrupación de clientes con las semejantes características y con un trato distinto.
- PT. Producto terminado
- Embarque de pedidos. Asignar producto del inventario a cada pedido.

#### *4.2.1.3 Referencias*

Diagramas de modelado UML.
Disponible online en: https://msdn.microsoft.com/es-es/library/dd409377.aspx
Obtenido el día 10 de noviembre de 2016

### 4.2.2 Arquitectura del sistema

#### *4.2.2.1 Casos de Uso*

##### 4.2.2.1.1 Diagrama General

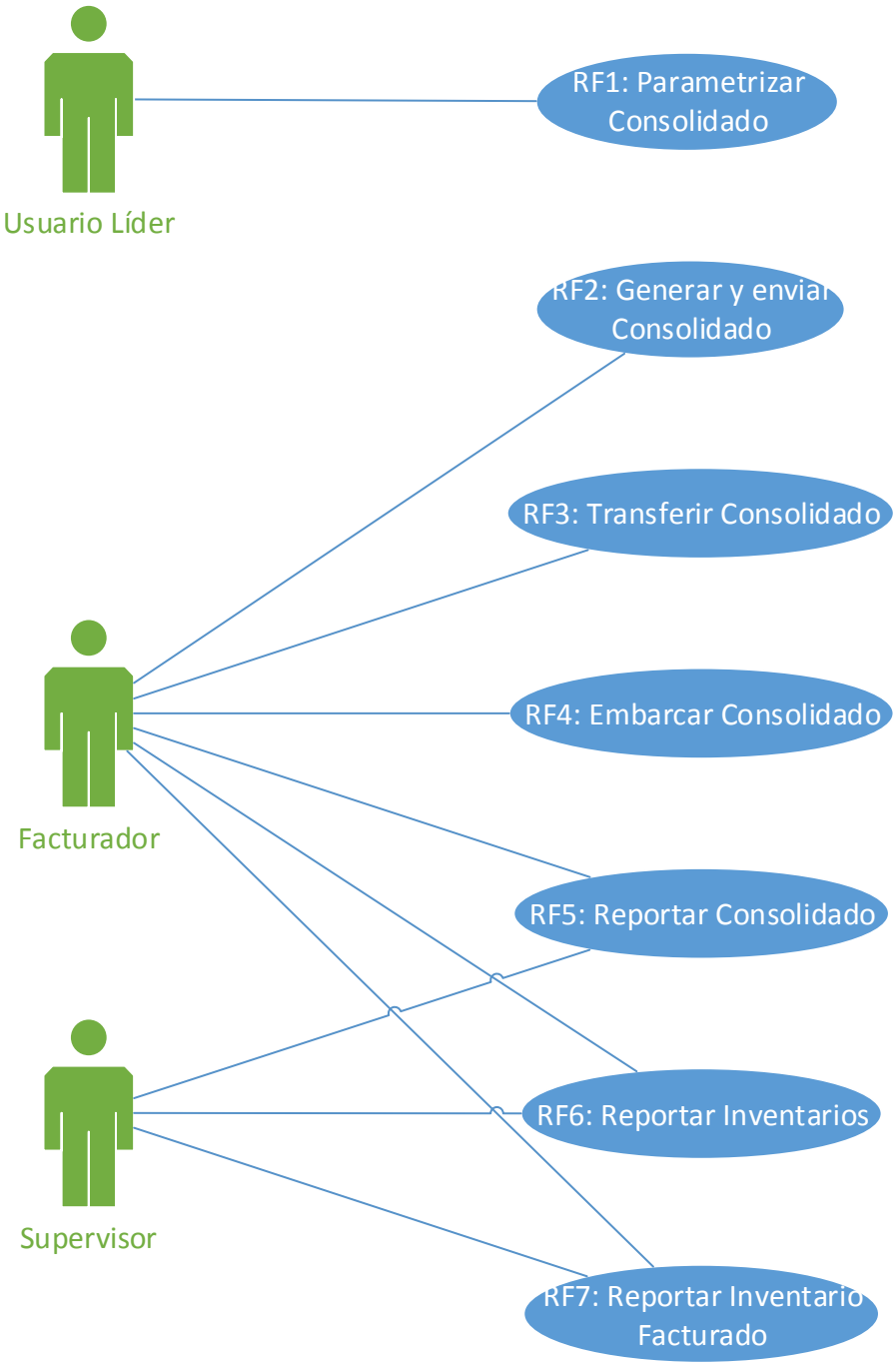


*Ilustración 0.4 Casos Usos General (Tobar J.; 2016)*

#### 4.2.2.1.2 RF1: Parametrizar Consolidado

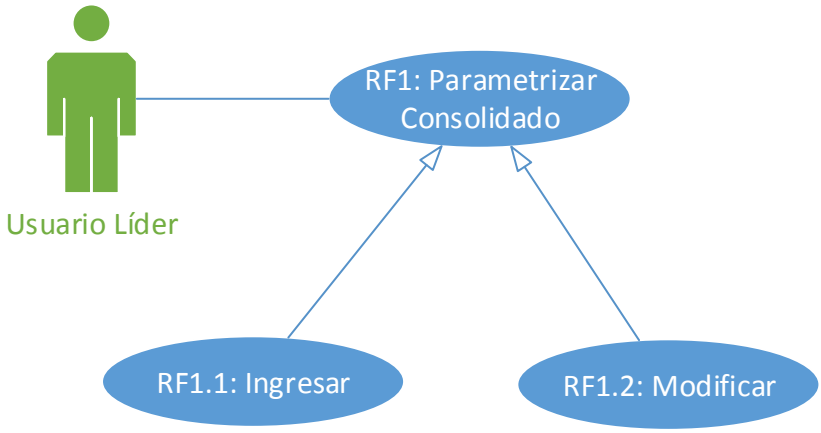


*Ilustración 0.5 RF1: Parametrizar Consolidado (Tobar J.; 2016)*

*RF1.1: Ingresar*

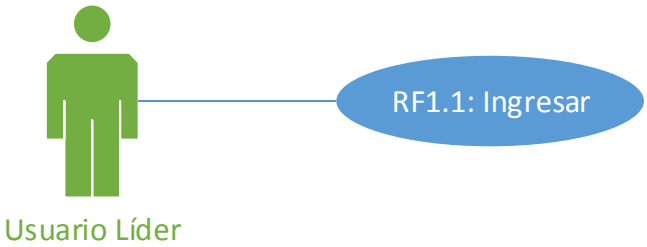


*Ilustración 0.6 RF1.1: Ingresar (Tobar J.; 2016)*

Descripción: El Usuario Líder podrá ingresar los almacenes que podrán crear los consolidados.

Actores: Usuario Líder.

Flujo Principal:

1. El actor ingresa la información del almacén de facturación.
2. El sistema verifica el almacén de facturación. (E1).
3. El sistema habilita los campos necesarios.
4. El actor presiona el botón siguiente.
5. El sistema guarda los datos.

Flujo Alterno

1. Si el almacén ya existe ir al caso RF1.2.

Excepciones:

*Tabla 0.5 RF1.1 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | El almacén de facturación no existe. | Ingresar el almacén nuevamente. |

*RF1.2: Modificar*

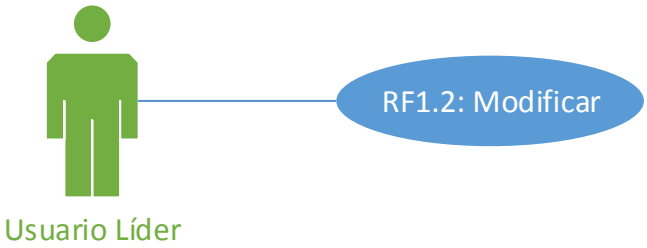


*Ilustración 0.7 RF1.2: Modificar (Tobar J.; 2016)*

Descripción: El supervisor podrá modificar los almacenes ya creados.

Actores: Supervisor.

Flujo Principal:

1. El actor ingresa la información del almacén de facturación que se desea modificar.
2. El sistema verifica el almacén de facturación. (E1).
3. El sistema habilita los campos necesarios.
4. El actor modifica los campos que desee.
5. El actor presiona el botón siguiente.
6. El sistema guarda los datos.

Excepciones:

*Tabla 0.6 RF1.2 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | El almacén de facturación no existe. | Ingresar el almacén nuevamente. |

#### 4.2.2.1.3 Generar y enviar Consolidado

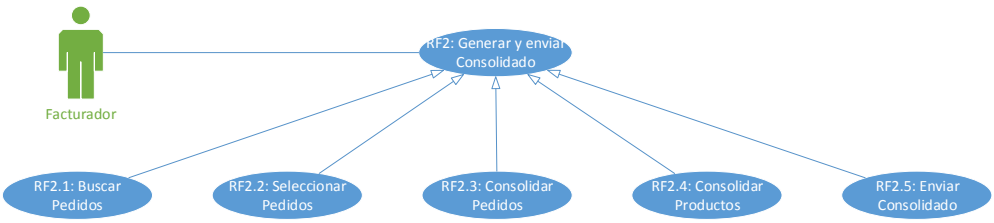


*Ilustración 0.8 RF1: Parametrizar consolidado (Tobar J.; 2016)*

*RF2.1: Buscar Pedidos*

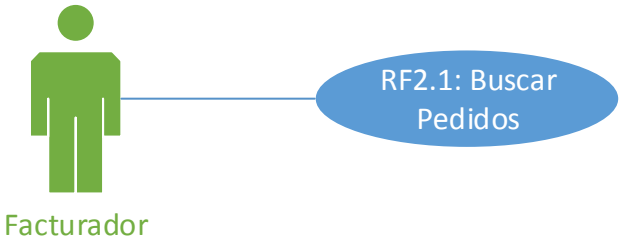


*Ilustración 0.9 RF2.1: Buscar Pedidos (Tobar J.; 2016)*

Descripción: El actor podrá buscar los pedidos que desea consolidar para la verificación del inventario.

Actores: Facturador.

Flujo Principal:

1. El actor ingresa el almacén.
2. El sistema valida el almacén. (E1).
3. El sistema habilita la búsqueda de pedidos.

4. El actor ingresa la información de los pedidos por rangos.

5. El sistema presenta un listado de los pedidos disponibles. (E2).

Excepciones:

*Tabla 0.7 RF2.1 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | El almacén de facturación no existe. | Ingresar el almacén nuevamente. |
| E2 | Los pedidos deben estar en estado LIB. | No presenta el pedido con estado LIB. |

*RF2.2 Seleccionar Pedidos*

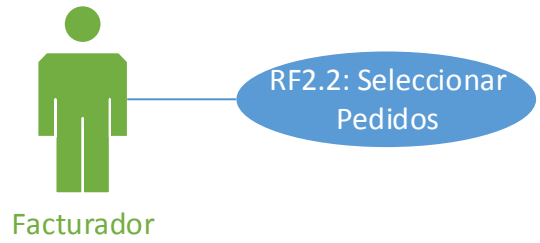


*Ilustración 0.10 RF2.2: Seleccionar Pedidos (Tobar J.; 2016)*

Descripción: El actor podrá seleccionar los pedidos que desea consolidar para la verificación del inventario.

Actores: Facturador.

Flujo Principal:

1. El actor selecciona los pedidos que desea consolidar.

2. El actor presiona siguiente.

*RF2.3: Consolidar Pedidos*

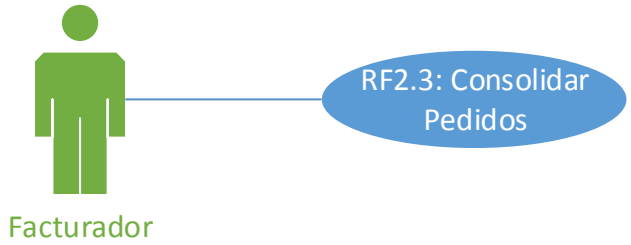


*Ilustración 0.11 RF2.3: Consolidar Pedidos (Tobar J.; 2016)*

Descripción: El actor podrá consolidar los pedidos.

Actores: Facturador.

Flujo Principal:

1. El sistema presenta información de los pedidos que desea consolidar.
2. El actor presiona consolidar pedidos seleccionados.

Flujo Alterno

1. El actor puede eliminar pedidos del consolidado.

*RF2.4: Consolidar Productos*

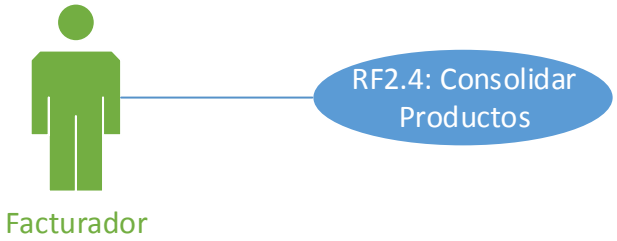


*Ilustración 0.12 RF2.4: Consolidar Productos (Tobar J.; 2016)*

Descripción: El actor podrá visualizar las cantidades que se van a solicitar al OPL, así como modificar los días de vencimiento de cada producto.

Actores: Facturador.

Flujo Principal:

1. El sistema presenta información de los productos y las cantidades pedidas, asignadas que desea consolidar. (E1).
2. El actor podrá cambiar los días de vencimiento de los productos que se requiera.
3. El actor presiona consolidar productos.

Excepciones:

*Tabla 0.8 RF2.4 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | La cantidad pedida supera al inventario. | La cantidad asignada será la cantidad existente en inventarios. |

*RF2.5 Enviar Consolidado*

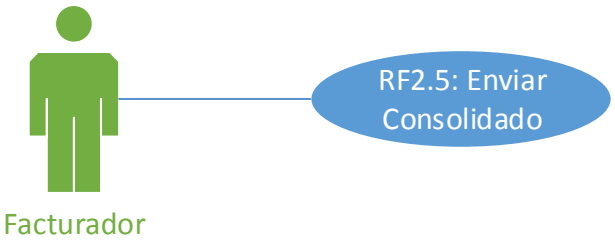


*Ilustración 0.13 RF2.5: Enviar Consolidado (Tobar J.; 2016)*

Descripción: El actor podrá guardar el consolidado y generar el archivo con las cantidades solicitadas.

Actores: Facturador.

Flujo Principal:

1. El actor presiona guardar consolidado.

2. El sistema almacenará los datos.
3. El sistema generará el archivo.

#### 4.2.2.1.4 Transferir Consolidados

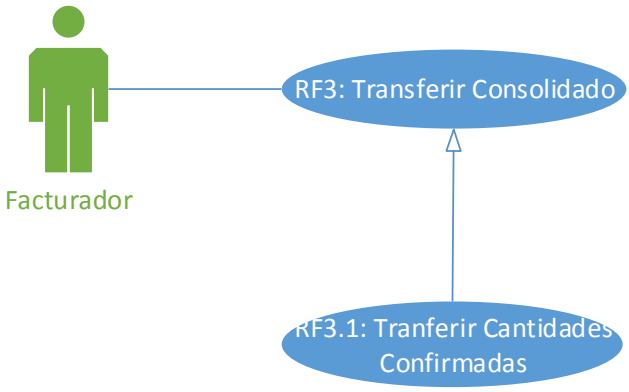


*Ilustración 0.14 Transferir Consolidado (Tobar J.; 2016)*

*RF3.1: Transferir Cantidades Confirmadas*

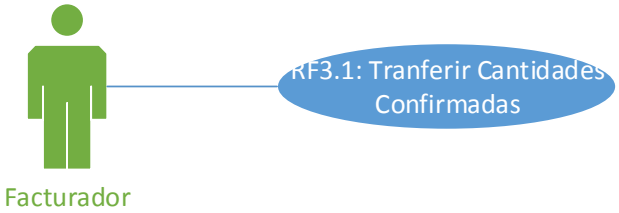


*Ilustración 0.15 RF3.1: Transferir Cantidades Confirmadas (Tobar J.; 2016)*

Descripción: El actor podrá transferir las cantidades que han sido confirmadas por el OPL.

Actores: Facturador.

Flujo Principal:

1. El actor ingresa el almacén y el consolidado.
2. El sistema valida el almacén y el consolidado. (E1).
3. El sistema obtendrá automáticamente la información confirmada por el OPL. (E2).
4. El actor presiona transferir consolidado.

5. El sistema transferirá las cantidades confirmadas por el OPL al almacén de facturación. (E3).

Excepciones:

*Tabla 0.9 RF3.1 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | El almacén de facturación y/o consolidado no existe. | Ingresar el almacén y/o consolidado nuevamente. |
| E2 | La información de los productos existe en la BD. | No carga la información de los productos que no existan en el inventario. |
| E3 | La cantidad a transferir no es disponible. | Solo se transferirán las cantidades que existen en el inventario. |

#### 4.2.2.1.5 Embarcar Consolidado

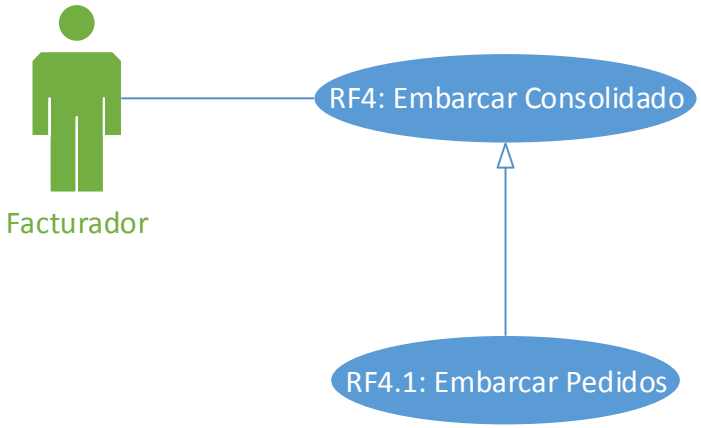


*Ilustración 0.16 Embarcar Consolidados (Tobar J.; 2016)*

*RF4.1: Distribuir Cantidades*


*Ilustración 0.17 RF4.1: Embarcar Pedidos (Tobar J.; 2016)*

Descripción: El actor podrá embarcar los pedidos con las cantidades que han sido confirmadas.

Actores: Facturador.

Flujo Principal:

1. El actor ingresa el consolidado.
2. El sistema valida el consolidado. (E1).
3. El actor presiona embarcar consolidados.
4. El sistema distribuye las cantidades consolidadas a cada uno de los pedidos que se encuentran consolidados.
5. El sistema embarca los productos a cada pedido que fueron consolidados.

Excepciones:

*Tabla 0.10 RF4.1 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | El consolidado no existe. | Ingresar consolidado nuevamente. |

#### 4.2.2.1.6 Reportar Consolidado


*Ilustración 0.18 Reportar Consolidado (Tobar J.; 2016)*

Descripción: El actor podrá visualizar los consolidados generados.

Actores: Facturador.

Flujo Principal:

1. El actor ingresa datos del consolidado.
2. El sistema valida los datos. (E1).
3. El actor selecciona el detalle que desea para los consolidados.
4. El actor presiona Siguiente.
5. El sistema presenta la información de los consolidados y sus detalles.

Excepciones:

*Tabla 0.11 RF5 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | Uno de los datos no existe. | Ingresar la información nuevamente. |

#### 4.2.2.1.7 Reportar Inventarios

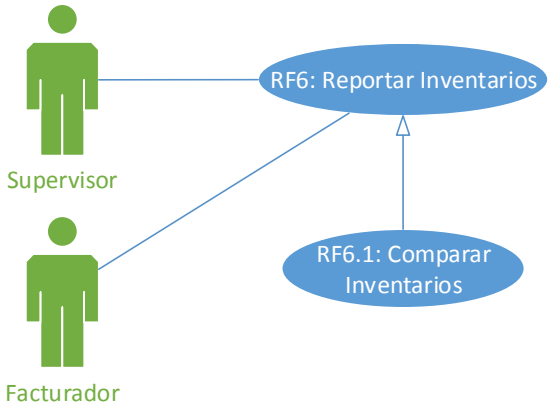


*Ilustración 0.19 Reportar Comparar Inventarios (Tobar J.; 2016)*

Descripción: El actor podrá visualizar las comparaciones de los inventarios.

Actores: Facturador.

Flujo Principal:

1. El actor ingresa datos del almacén.
2. El sistema valida los datos. (E1).
3. El actor presiona Siguiente.
4. El sistema obtiene la información del inventario del OPL.
5. El sistema compara los inventarios.
6. El sistema presenta la diferencia de inventarios.

Excepciones:

*Tabla 0.12 RF6.1 Excepciones*

| Excepción | Motivo | Como Solucionar |
|---|---|---|
| E1 | El almacén no existe. | Ingresar el almacén nuevamente. |

### *4.2.2.2 Diagramas de base de datos*

Se crearon las siguientes tablas:

- acon_hist: Almacenes que serán permitidos consolidar.
- docto_hist: Cabecera de consolidados.
- doctod_det: Detalle de productos de consolidados.
- doctop_hist: Histórico de pedidos que conforman el consolidado.

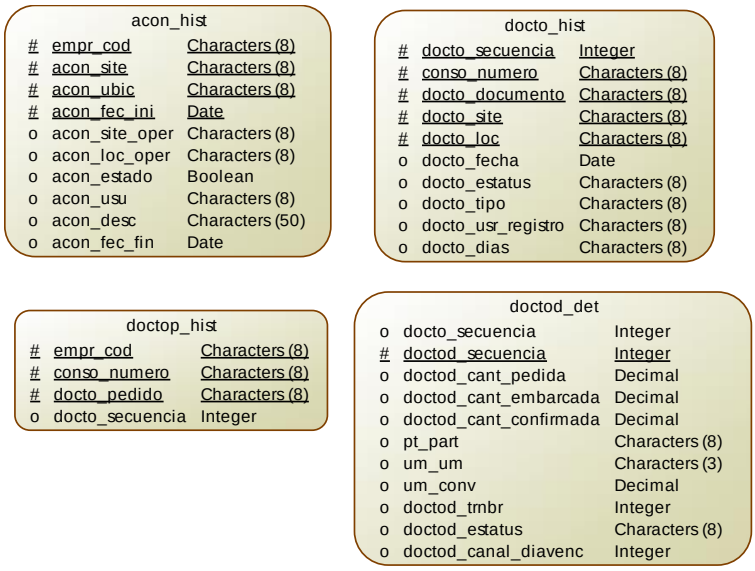


*Ilustración 0.20 Tablas agregadas (Tobar J.; 2016)*

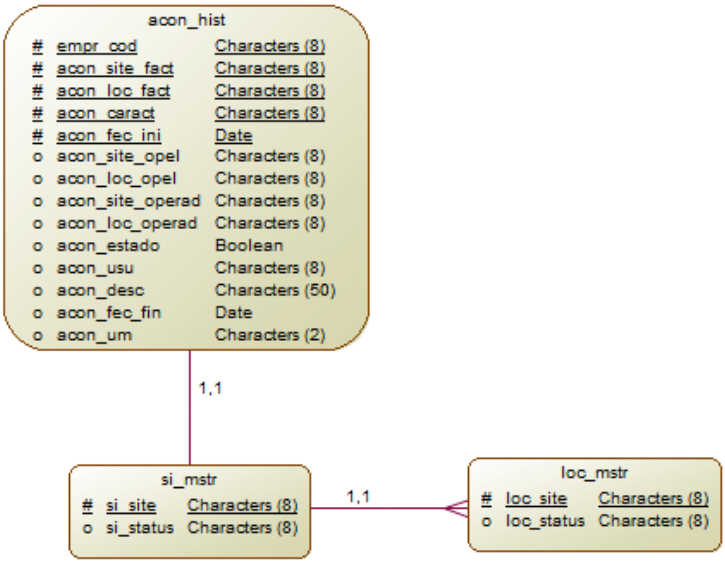


*Ilustración 0.21 Diagrama Parametrizar Consolidados (Tobar J.; 2016)*

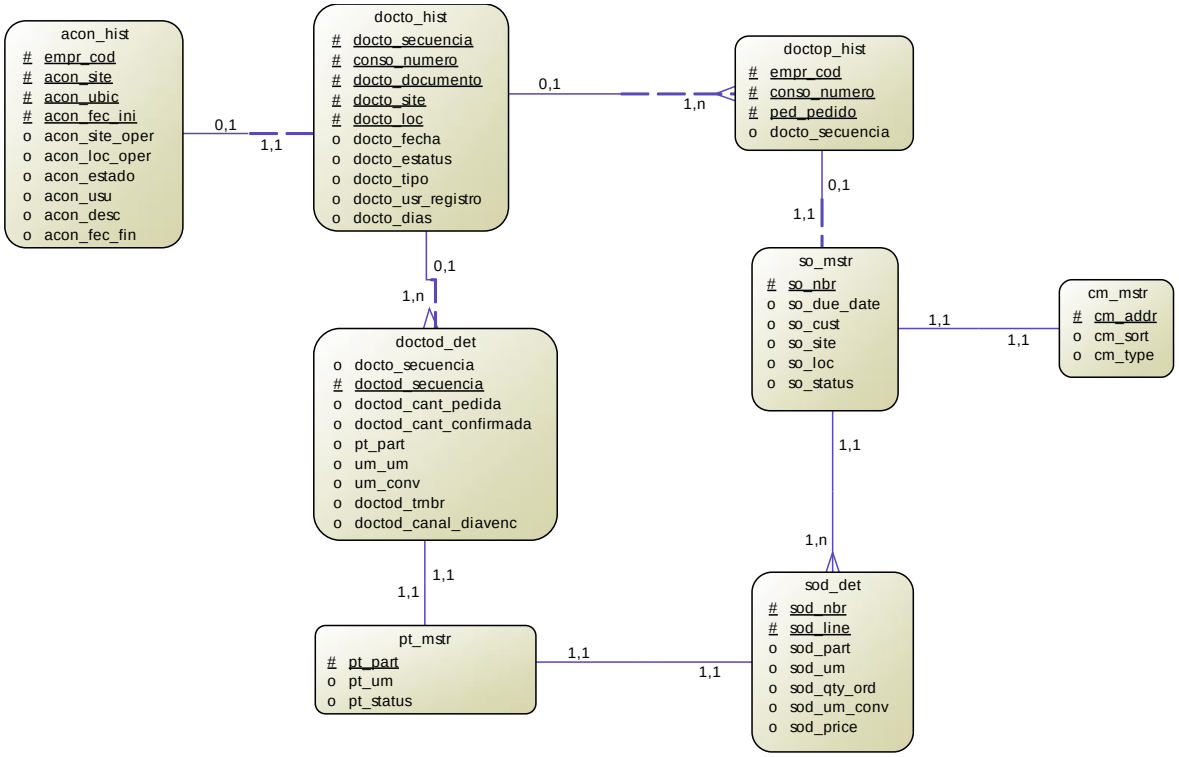


*Ilustración 0.22 Diagrama Generar y enviar Consolidado (Tobar J.; 2016)*

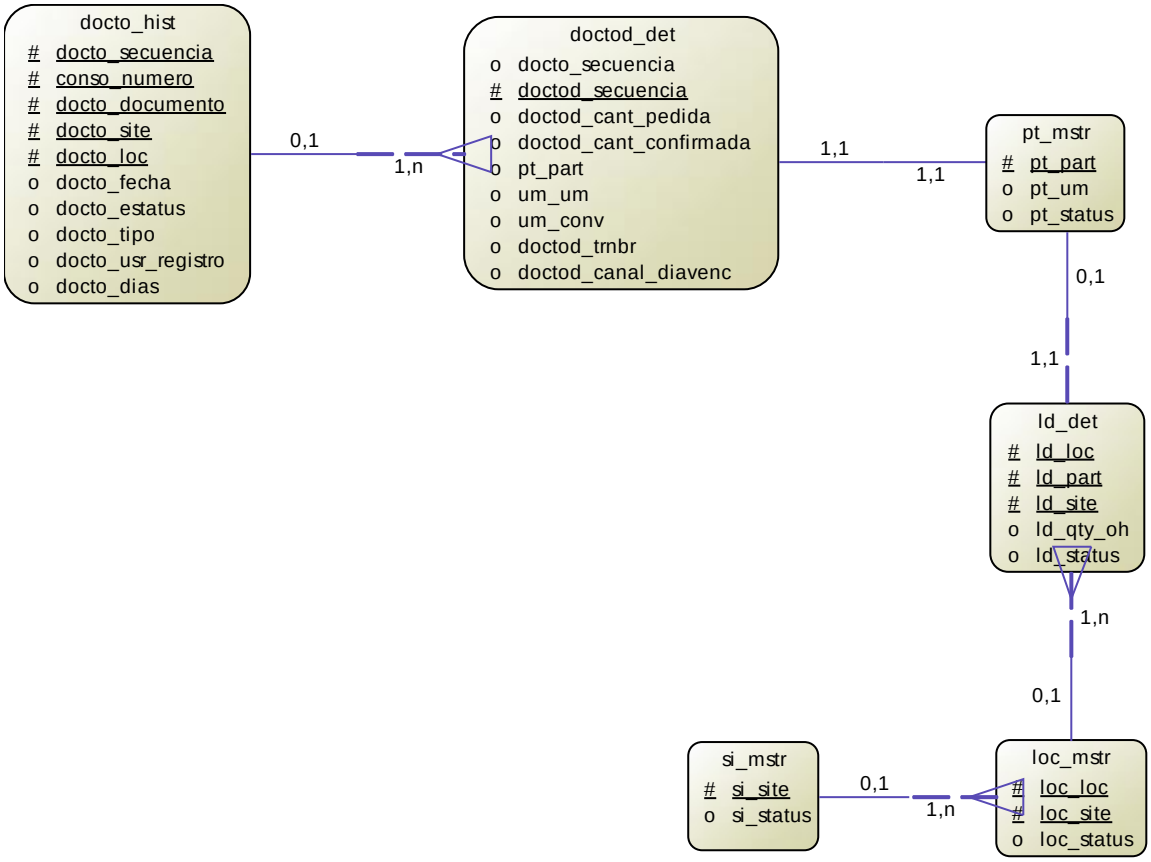


*Ilustración 0.23 Diagrama Transferir Consolidado (Tobar J.; 2016)*

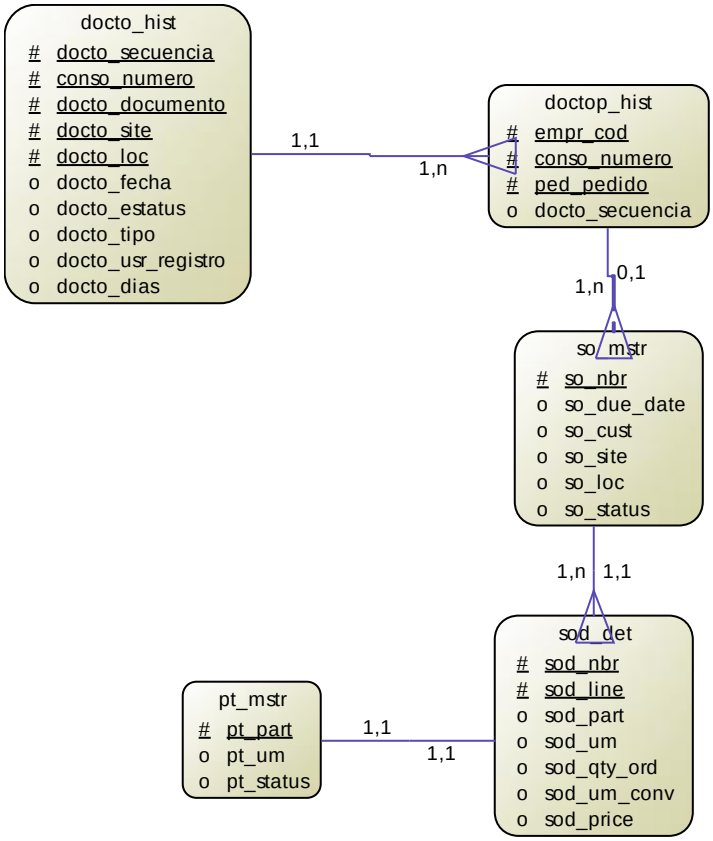


*Ilustración 0.24 Diagrama Embarcar Consolidado (Tobar J.; 2016)*

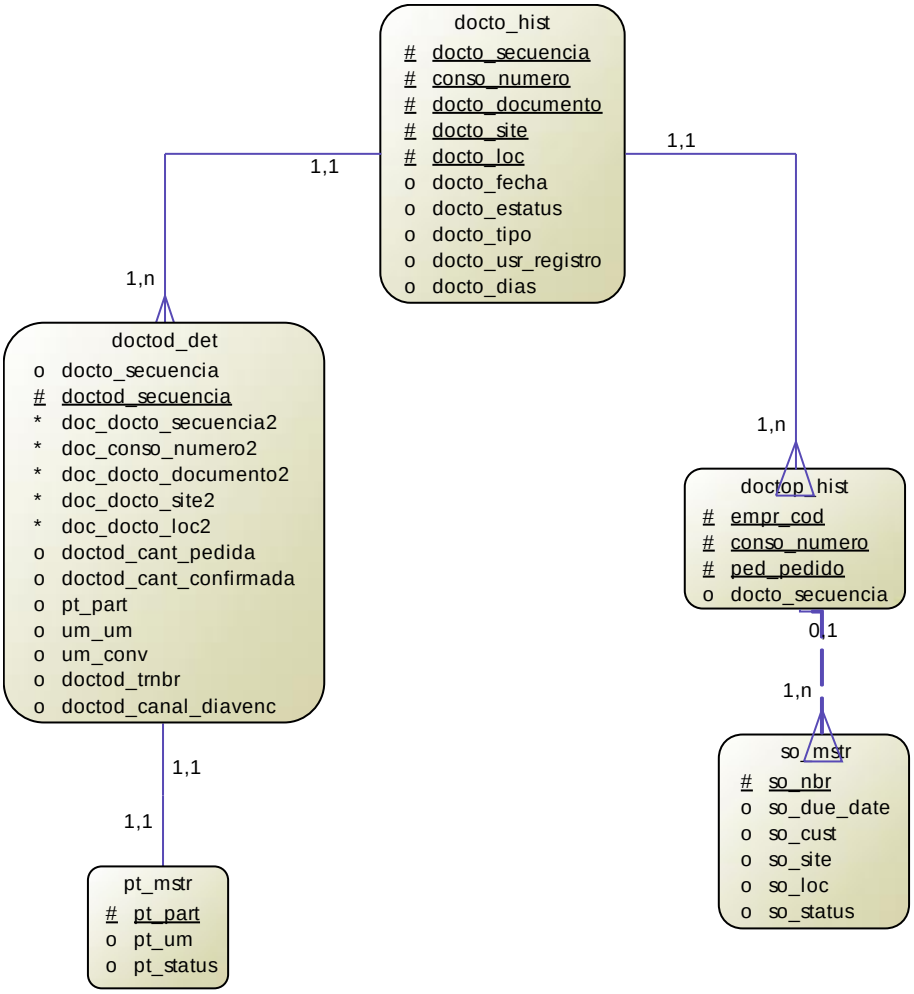


*Ilustración 0.25 Diagrama Reportar Consolidado (Tobar J.; 2016)*

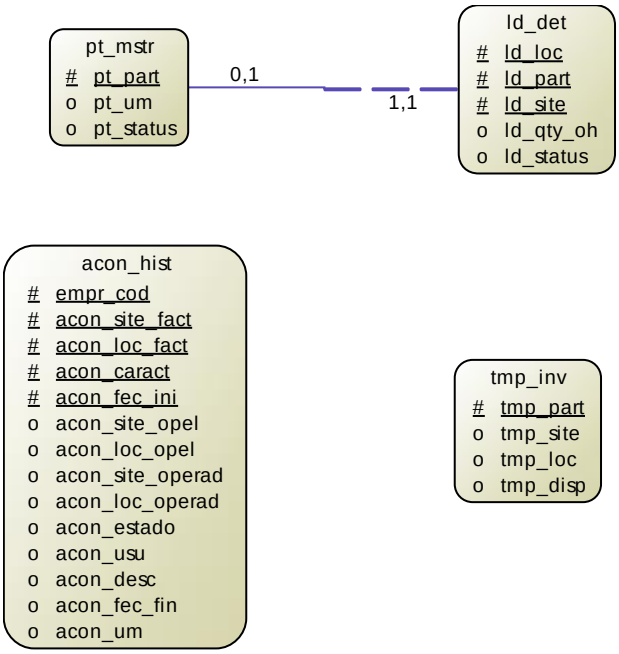


*Ilustración 0.26 Diagrama Comparar Inventario (Tobar J.; 2016)*

### *4.2.2.3 Diagramas de secuencias*

#### 4.2.2.3.1 Generar Consolidado

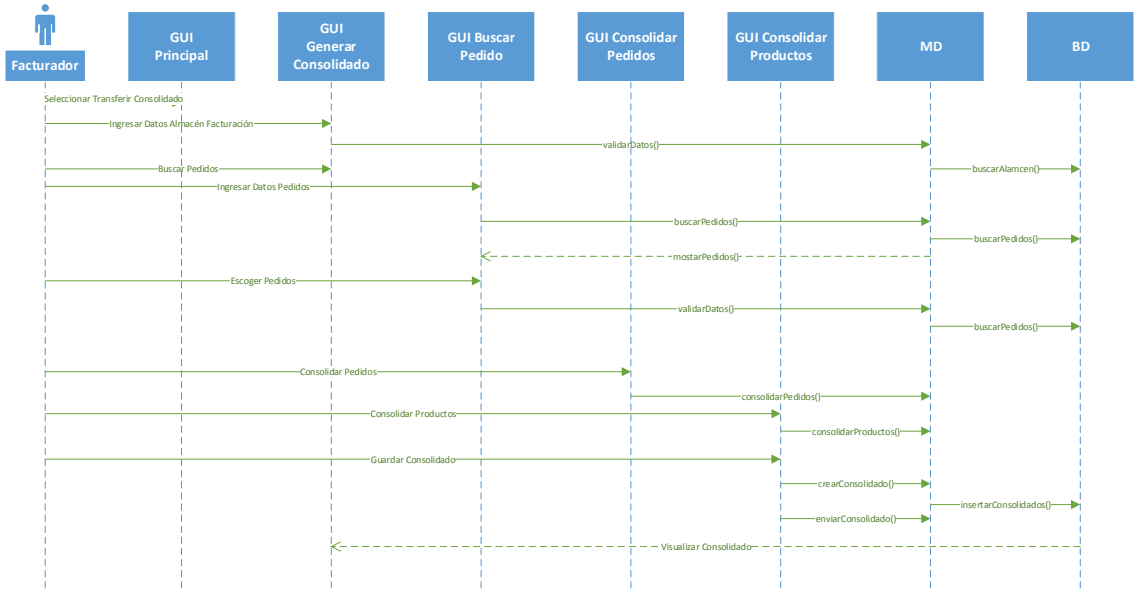


*Ilustración 0.27 Secuencia Generar Consolidado (Tobar J.; 2016)*

#### 4.2.2.3.2 Transferir Consolidados

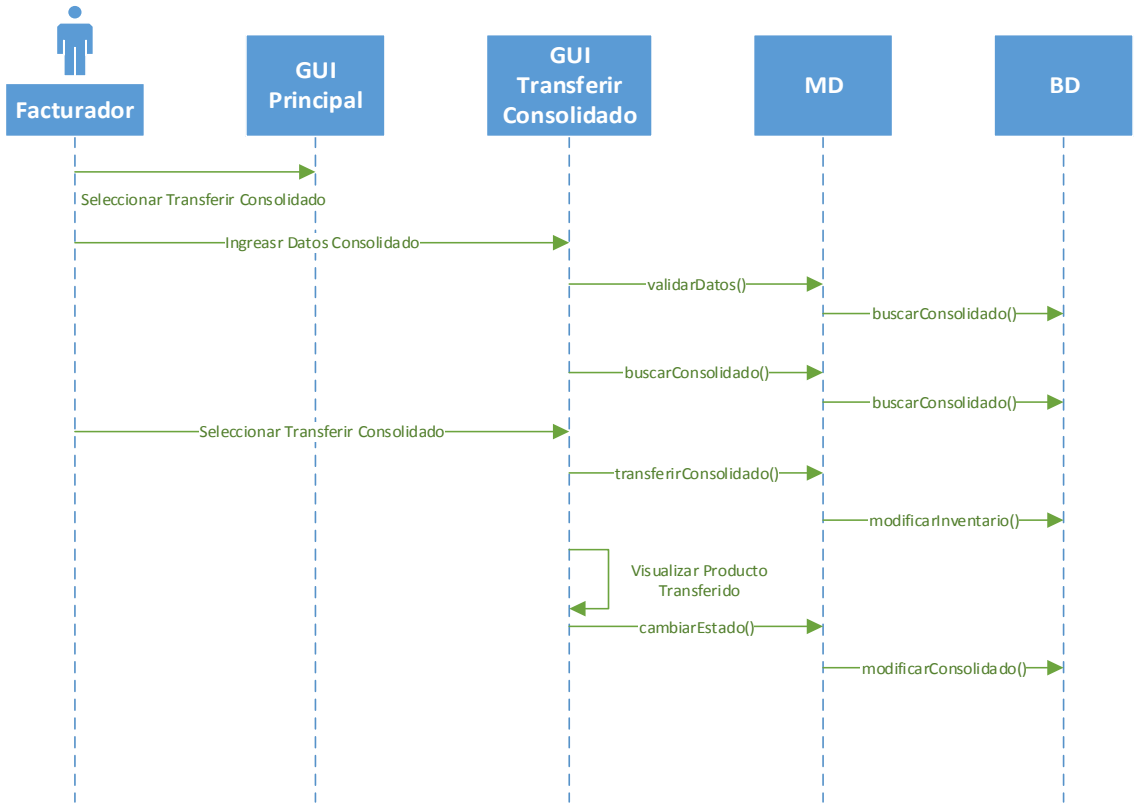


*Ilustración 0.28 Secuencia Transferir Consolidados (Tobar J.; 2016)*

#### 4.2.2.3.3 Embarcar Consolidados

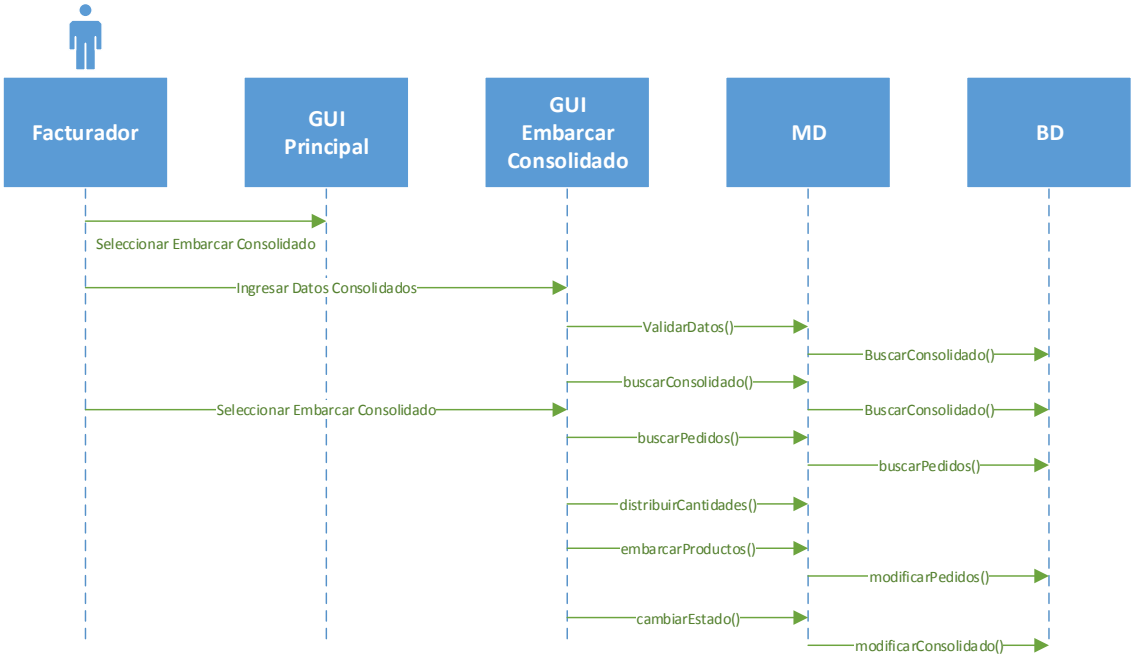


*Ilustración 0.29 Secuencia Embarcar Consolidado (Tobar J.; 2016)*

#### 4.2.2.3.4 Reporte Inventarios

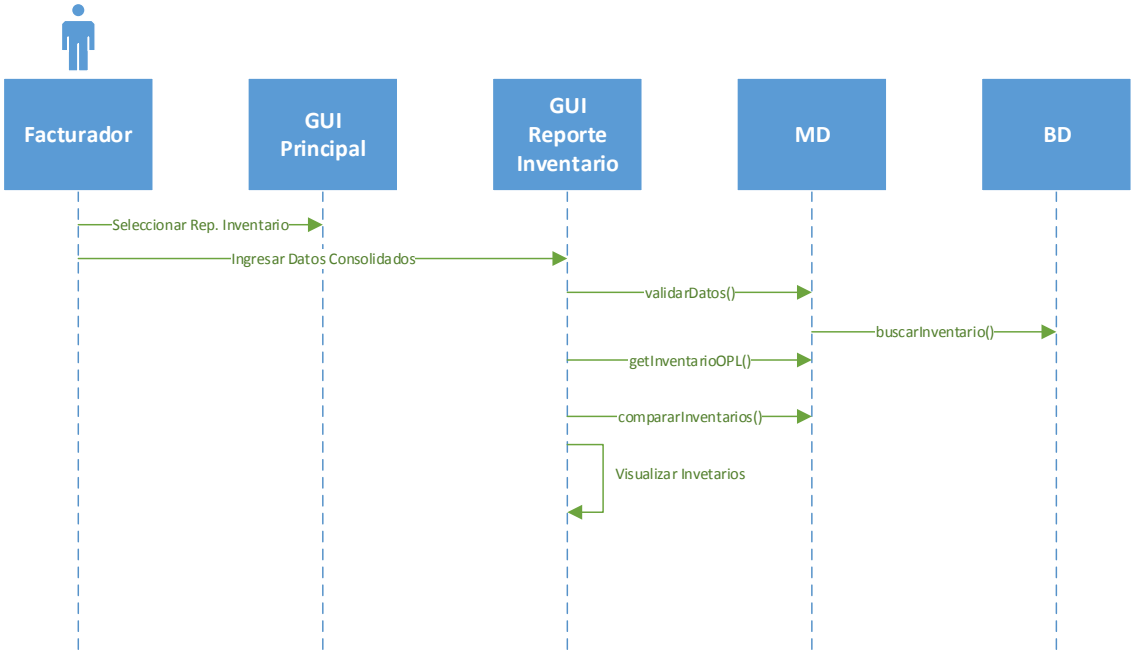


*Ilustración 0.30 Secuencia Reporte Inventario (Tobar J.; 2016)*

#### 4.2.2.3.5 Reporte Consolidados

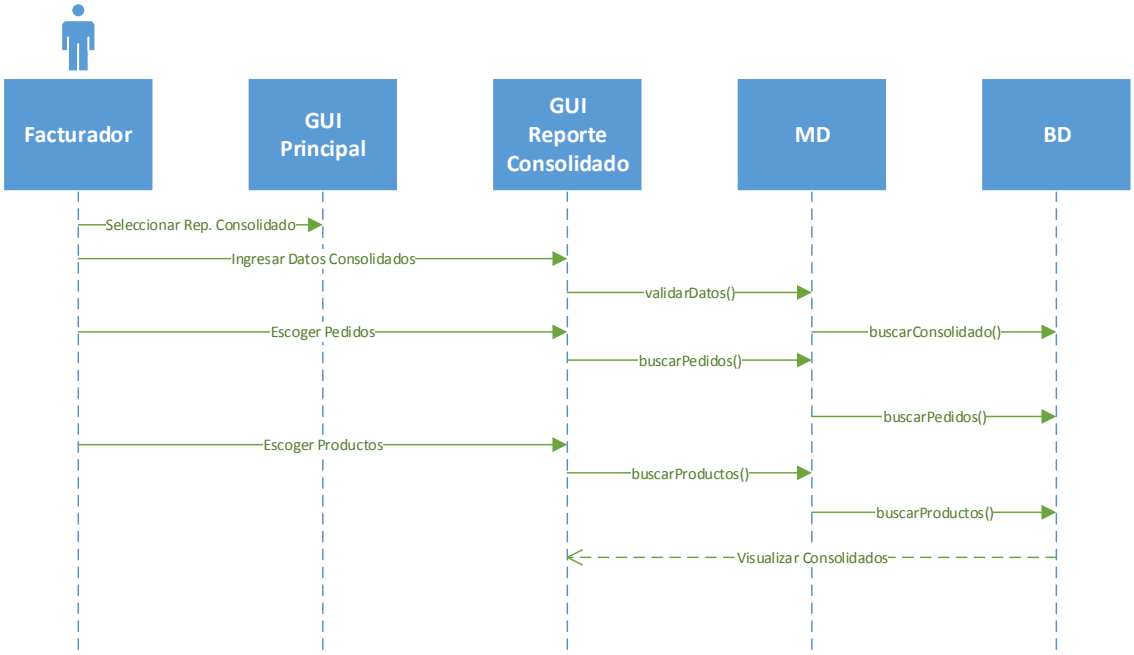


*Ilustración 0.31 Secuencia Reporte Consolidado (Tobar J.; 2016)*

## 4.3 Pruebas.

*Tabla 13 Pruebas (Tobar J.; 2016)*

| **RF** | **Caso de uso** | **Eventos** | **Acciones a probar** |
|---|---|---|---|
| RF1 | Parametrizar consolidados | Ingresar | Validar el nuevo almacén de facturación que se ingresó |
| | | | El sistema habilita campos a llenar |
| | | Modificar | Validar que exista el almacén que se va a modificar |
| | | | Modificar la información parametrizada |
| RF2 | Generar y enviar Consolidados | Buscar Pedidos | Validar el almacén |
| | | | El sistema presenta la lista de pedidos que cumplen con los filtros que se ingresaron |

| | | | |
|---|---|---|---|
| | | Seleccionar pedidos | Se puede seleccionar algunos pedidos |
| | | | El sistema solo presentará los pedidos que están listos para ser facturados |
| | | Consolidar pedidos | El sistema consolida los pedidos seleccionados |
| | | | Se puede eliminar los pedidos que fueron seleccionados |
| | | Consolidar productos | El sistema presenta la lista de productos de los pedidos que fueron consolidados |
| | | | Se puede modificar la fecha de caducidad de cada producto |
| | | Enviar consolidado | Se guarda el consolidado |
| | | | El sistema generará el archivo con la información del consolidado generado |
| RF3 | Transferir consolidados | Transferir cantidades confirmadas | Validar que existan los almacenes entre los cuales se van a realizar las transferencias |
| | | | Validar que exista el consolidado que se desea transferir |
| | | | Validar que el sistema transfiera únicamente las cantidades que fueron confirmadas por el OPL |
| | | | El sistema no transferirá productos que no existan en el inventario |
| RF4 | Embarcar | Embarcar | Validar que exista el consolidado ingresado |

| | | | |
|---|---|---|---|
| | consolidado | pedidos | El sistema debe distribuir las cantidades confirmadas para los pedidos consolidados |
| RF5 | Reportar consolidados | Detalle pedidos | Validar la información de los almacenes ingresados |
| | | | El sistema debe presentar la información de los pedidos que han sido consolidados |
| | | Detalle productos | Validar la información de los almacenes ingresados |
| | | | El sistema debe presentar las cantidades de los productos que han sido consolidados |
| | | General | Validar la información de los almacenes ingresados |
| | | | El sistema debe presentar la información de los consolidados |
| RF6 | Reportar inventarios | Reportar | Validar la información de los almacenes ingresados |
| | | | Comparar los inventarios de las dos empresas |
| | | Obtener información | El sistema debe obtener la información enviada por el OPL |

## 4.4 Implementación

Se anexan los manuales de usuario e instalación.

# Capítulo 5: Conclusiones y Recomendaciones

## 5.1 Conclusiones

- Al momento de diseñar una interface con otra empresa se deben crear opciones que permitan parametrizar la funcionalidad según la necesidad.
- Para realizar un buen levantamiento de información es importante contar con usuarios que tengan la suficiente experticia para poder identificar los requerimientos necesarios para satisfacer sus necesidades.
- Cuando se definan los requerimientos funcionales y no funcionales es necesario acompañar al usuario para definir los requerimientos, y así se podrá identificar el tipo de requerimiento.
- Al realizar el diseño de la solución se debe basar en los requerimientos del usuario que son definidos en el SRS.
- La metodología y lenguaje de programación que se van a utilizar para el desarrollo de software va a depender del proyecto, no se puede utilizar una metodología y lenguaje en especial, ya que eso depende del entorno, características y las restricciones del proyecto.
- Las pruebas que el usuario debe realizar se las define en base a los casos de uso, ya que ahí se puede observar la interacción del usuario con el sistema.
- Contar con un usuario es necesario al momento de realizar el manual de usuario, ya que el usuario nos podrá apoyar en la manera de explicar la usabilidad del software.

## 5.2 Recomendaciones

- Antes de automatizar un proceso se debe asegurar que se lo sistematice, de esta manera se podrá conocer los verdaderos requerimientos.
- Mantener una permanente comunicación con el usuario permitirá al usuario visualizar el avance del proyecto y en caso de existir cambio se los podrá atender de una manera más ágil.
- No existe una metodología de desarrollo de software perfecta, se debe identificar si el entorno es cambiante o regular, según el tipo de entorno escoger la metodología ágil o tradicional.
- Al realizar el diseño de la solución el desarrollador debe ponerse en el lugar del usuario final, para desarrollar un software que sea amigable al usuario.
- La fase más importante del desarrollo de software es el levantamiento de información, ya que en esta fase en conjunto con el usuario se van a definir las características del software.
- Las pruebas realizadas por el usuario deben ser exhaustivas y realizadas por un usuario con la suficiente experticia para verificar que todas las nuevas funcionalidades cumplan los requerimientos definidos.

# Bibliografía

*Agile Alliance*. (16 de 10 de 2016). Obtenido de https://www.agilealliance.org/agile101/the-agile-manifesto/

*Agile Manifesto*. (16 de 10 de 2016). Obtenido de http://agilemanifesto.org/iso/es/principles.html

*Ambysoft*. (23 de 10 de 2016). Obtenido de http://www.ambysoft.com/unifiedprocess/agileUP.html

Cabot Sagrera, J. (2013). *Ingeniería del software.* Barcelona: Ediorial UOC.

Canales Mora, R. (2010). *Informática profesional: las reglas no escritas para triunfar en la empresa (2a. ed.).* Madrid: RA-MA Editorial.

Casado Iglesias, C. (2014). *Entornos de desarrollo.* Madrid: RA-MA Editorial.

Lajara Vizcaíno, J. R. (2007). *LabVIEW: entorno gráfico de programación.* Barcelona: Marcombo.

López Goytia, J. L. (2014). *Programación orientada a objetos C++ y Java: un acercamiento interdisciplinario.* México D.F.: Grupo Editorial Patria.

*Metodologías Agiles*. (23 de 10 de 2016). Obtenido de https://metodologiasagiles.wikispaces.com/metodos+agiles+vs+metodos+tradicionales

MSDN. (2016). *Diagramas de modelado UML*. Obtenido de https://msdn.microsoft.com/es-es/library/dd409377.aspx

Noguera Otero, F. J., & Riera Terrén, D. (2013). *Programación.* Barcelona: UOC.

*Osl2*. (23 de 10 de 2016). Obtenido de http://osl2.uca.es/wikiCE/index.php/Archivo:ComparacionXP.PNG

*Progress*. (23 de 10 de 2016). Obtenido de https://www.progress.com/docs/default-source/openedge/progressovrw_openedge_abl.pdf?sfvrsn=2

*Progress Community*. (23 de 10 de 2016). Obtenido de https://community.progress.com/community_groups/openedge_general/w/openedgegeneral/2743.openedge-11-6-product-documentation

*Progress Documentation*. (23 de 10 de 2016). Obtenido de http://documentation.progress.com/output/OpenEdge/wwhelp/wwhimpl/common/html/wwhelp.htm?context=dvhbk&file=dvhbk-01-1.html

*Progress OpenEdge*. (16 de 10 de 2016). Obtenido de https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/gsins/preface.html#

Teniente López, E., Costal Costa, D., & Sancho Samsó, R. (2013). *Especificación de sistemas software en UML*. Barcelona: Universitat Politècnica de Catalunya.

*Ubuntu Adempiere*. (23 de 10 de 2016). Obtenido de http://ubuntu-adempiere.blogspot.com/2011/09/metodologia-aup-agile-unified-process.html

Villada Romero, J. L. (2015). *Desarrollo y optimización de componentes software para tareas administrativas de sistemas.* Málaga: IC Editorial.

*Wikipedia*. (23 de 10 de 2016). Obtenido de https://es.wikipedia.org/wiki/Archivo:Esquema_general_de_una_metodologia_agil_para_desarrollo_de_software.png