

**UNIVERSIDAD DE CASTILLA-LA MANCHA ESCUELA SUPERIOR DE INFORMÁTICA**

**INGENIERÍA**

**EN INFORMÁTICA**

**PROYECTO FIN DE CARRERA**

*“Línea de productos software: Juegos de mesa”*

Juan Yánez García-Catalán

**Septiembre, 2012**



**UNIVERSIDAD DE CASTILLA-LA MANCHA ESCUELA SUPERIOR DE INFORMÁTICA** (Departamento de Tecnologías y Sistemas de Información)

**PROYECTO FIN DE CARRERA**

*“Línea de productos software: Juegos de mesa”*

Autor: Juan Yánez García-Catalán

Director: Macario Polo Usaola

**Septiembre, 2012**

**TRIBUNAL:**

**Presidente:   
  
Vocal 1:   
  
Vocal 2:**

**Secretario:**

**FECHA DE DEFENSA:**

**CALIFICACIÓN:**

**PRESIDENTE VOCAL 1 VOCAL 2 SECRETARIO**

Fdo.: Fdo.: Fdo.: Fdo.:

## RESUMEN

Resumen

## ABSTRACT

Resumen en ingles.

## ÍNDICE GENERAL

[RESUMEN I](#_Toc332937538)

[ABSTRACT II](#_Toc332937539)

[ÍNDICE GENERAL V](#_Toc332937540)

[ÍNDICE DE FIGURAS VI](#_Toc332937541)

[1. INTRODUCCIÓN 1](#_Toc332937542)

[1.1. Introducción al tema 1](#_Toc332937543)

[1.2. Estructura del documento 4](#_Toc332937544)

[2. OBJETIVOS DEL PROYECTO 7](#_Toc332937545)

[2.1. Objetivo principal 7](#_Toc332937546)

[2.2. Objetivo secundario 7](#_Toc332937547)

[2.3. Herramientas y medios empleados 7](#_Toc332937548)

[2.3.1. Middleware de comunicaciones 7](#_Toc332937549)

[2.3.2. Lenguaje de programación 8](#_Toc332937550)

[2.3.3. Entorno de desarrollo 8](#_Toc332937551)

[2.3.4. Base de datos 9](#_Toc332937552)

[2.3.5. Herramienta de análisis y diseño 9](#_Toc332937553)

[2.3.6. Herramientas de gestión 9](#_Toc332937554)

[3. ESTADO DE LA CUESTIÓN 11](#_Toc332937555)

[3.1. Introducción 11](#_Toc332937556)

[3.1.1. Reutilización 12](#_Toc332937557)

[3.2. Líneas de productos software 13](#_Toc332937558)

[3.2.1. Modelado de la variabilidad 14](#_Toc332937559)

[3.2.2. Introducción al desarrollo basado en UML 17](#_Toc332937560)

[3.3. Ingeniería de líneas de productos software 17](#_Toc332937561)

[3.3.1. Modelo de proceso evolutivo 17](#_Toc332937563)

[3.3.2. Fases del proceso de Ingeniería en líneas de producto software 19](#_Toc332937564)

[3.3.3. Enfoques de construcción de una línea de productos software 22](#_Toc332937565)

[4. MÉTODO DE TRABAJO 26](#_Toc332937566)

[4.1. PLUS: Ingeniería de Software para Líneas de Productos basadas en UML 26](#_Toc332937567)

[4.1.1. Proceso Unificado de Desarrollo Software 26](#_Toc332937568)

[4.1.2. Integración de PLUS con el Proceso Unificado de Desarrollo 30](#_Toc332937569)

[4.1.3. Ingeniería de producto 32](#_Toc332937570)

[5. RESULTADOS 35](#_Toc332937571)

[5.1. Ingeniería de dominio 35](#_Toc332937572)

[5.1.1. Inicio 35](#_Toc332937573)

[5.2. Ingeniería de producto 35](#_Toc332937574)

[6. CONCLUSIONES Y PROPUESTAS 37](#_Toc332937575)

[BIBLIOGRAFÍA 39](#_Toc332937576)

## ÍNDICE DE FIGURAS

[Figura 1.1 Modelo básico de una línea de productos software 3](#_Toc332925263)

[Figura 3.1 Ejemplo de línea de productos en la historia antigua 11](#_Toc332925264)

[Figura 3.2 Modelo de proceso evolutivo 19](#_Toc332925265)

[Figura 3.3 Ingeniería de la línea de productos software. 20](#_Toc332925266)

[Figura 4.1 Evolución de la arquitectura del sistema 27](#_Toc332925267)

[Figura 4.2 Iteración en el Proceso Unificado de Desarrollo 29](#_Toc332925268)

[Figura 4.3 Esfuerzo en las actividades según la fase del proyecto 30](#_Toc332925269)

## INTRODUCCIÓN

En este capítulo, se introduce la problemática en la que se centra el Proyecto, perfilando una posible solución. Además se explica el contexto donde se engloba el proyecto y las relaciones de éste con el entorno en el que se desarrolla.

### Introducción al tema

Cuando una empresa ofrece un producto software a distintos clientes surge toda la problemática de las versiones y la evolución acompasada del producto. En este escenario, no es raro encontrarse con alguna de las siguientes situaciones (1):

* **Relaciones conflictivas entre los equipos de marketing y de ingeniería** a causa de la incapacidad de los segundos de adecuarse a las solicitudes de nuevas variaciones de producto provenientes del negocio.
* **Coordinación compleja y costosa** de múltiples tareas de desarrollo en paralelo que comparten software común.
* **Código fuente poco robusto**, que resulta difícil de extender con variaciones del nuevo producto propenso al error.

**La producción en serie** (*mass production*) –la capacidad para crear eficientemente múltiples copias del mismo producto– constituyó un gran avance en el mundo de la fabricación. Las líneas de producto llevan años aplicándose en otros entornos industriales, como la fabricación de vehículos en cadenas de montaje: las diferentes versiones de un mismo modelo de coche difieren en ciertas características de su equipamiento (presencia o ausencia de aire acondicionado, elevalunas eléctricos, etc.), lo cual no impide que todos ellos compartan las mismas cadenas de montaje y producción. Más recientemente, las líneas de producto se utilizan también para el desarrollo y construcción de teléfonos móviles que, además del conjunto básico de funcionalidades, ofrecen diferentes tamaños de pantalla, presencia o ausencia de bluetooth, varias resoluciones de la cámara, etc.

Pero crear múltiples copias de un producto software es trivial. Sin embargo, **la personalización en serie** (*mass customization*) –la capacidad para crear eficientemente múltiples variaciones de un producto– es un importante reto tanto en la fabricación de lavadoras como en la venta de un ERP o cualquier otro producto software. La pregunta es cómo se plasma el enfoque de personalización en serie en el desarrollo de productos software.

En la producción de software, el proceso ha venido estando centrado en el producto antes que en la línea de montaje. Las herramientas de desarrollo (IDE) y las metodologías ayudaban a agilizar y sistematizar la creación de un único producto. Sí que existe una inquietud por reutilizar pero en la mayor parte de los casos, la reutilización es oportunista, es decir, surgía la posibilidad de reutilizar a posteriori, no era algo que se supiera positivamente que se iba a poder reutilizar. Por ello, muchos esfuerzos de re-utilización no se amortizaban ya que no terminaba de surgir la oportunidad para poder reutilizarlo.

Esta situación cambia con las líneas de producto. Ahora ya no se quiere producir un único producto, sino una cadena de montaje que gestione eficiente y eficazmente las diferentes variaciones que pueden existir entre los productos. La empresa ya no se centra en un producto para un cliente (por ejemplo, construir un portal para Iberia), sino en un dominio (por ejemplo, construir portales para líneas aéreas). El reto está en delimitar el ámbito de este dominio, identificar las variaciones que se van a soportar, y dotarse de la infraestructura que permita producir el producto a bajo coste pero manteniendo altas cotas de calidad. Es decir, aplicar los principios de la producción en serie también al software. Este enfoque resulta en mejoras tanto en la eficiencia (reducción del time-to-market) como en la eficacia (mejora de la calidad del software).

**Una línea de productos software (*LPS*)** se define por tanto como “*un conjunto de sistemas software, que comparten un conjunto común de características (features), las cuales satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base (core assets) de una manera preestablecida*” (2). Uno de los aspectos distintivos de las LPS frente al desarrollo tradicional es la importancia de la variabilidad a lo largo de todo el proceso de desarrollo: los productos de la línea comparten un **conjunto de características** (*commonalities*) y difieren en determinados **puntos de variación** (*variation points*), que representan la variabilidad entre los productos.

Un aspecto central en el desarrollo de LPS es la división de los procesos de ingeniería: la Ingeniería de Dominio, responsable de desarrollar los elementos comunes a todos los integrantes de la línea y su mecanismo de variabilidad, y la Ingeniería del Producto (o Ingeniería de la Aplicación), cuyos cometidos incluyen desarrollar los productos para clientes concretos, reutilizando los recursos creados en la Ingeniería del Dominio (véase figura 1.1).



Figura 1.1 Modelo básico de una línea de productos software

Además existen dos enfoques para la construcción de los productos de la LPS:

* **Enfoque sustractivo**: Todas las variabilidades se contemplan en el nivel de ingeniería de dominio y, a nivel de producto, se eliminan aquellas que no corresponden al producto concreto que se está desarrollando.
* **Enfoque aditivo:** A nivel de ingeniería de dominio se trabaja únicamente con las *common features*. Luego, a nivel de producto, se le incorporan las variabilidades que correspondan.

### Estructura del documento

El presente documento ha sido redactado siguiendo la normativa del proyecto fin de carrera de la Escuela Superior de Informática de la universidad de Castilla-La-Mancha. Éste consta de los siguientes capítulos:

**Capítulo 1. Introducción**

En este capítulo se ha introducido la problemática en la que se centra este proyecto fin de carrera, dando unas pinceladas de que se pretende resolver. Además se explica el contexto donde se engloba y las relaciones de éste con el entorno en que se desarrolla.

**Capítulo 2. Objetivos**

En este capítulo se hablará de los objetivos que se pretenden conseguir en el desarrollo de este proyecto fin de carrera y las herramientas necesarias para llevarlo a cabo.

**Capítulo 3. Antecedentes**

.

**Capítulo 4. Método de trabajo**

En este capítulo se presenta el método de trabajo seguido para el desarrollo de la Línea de productos software. Se explicará la metodología de desarrollo para el análisis, diseño e implementación de la aplicación

**Capítulo 5. Resultados**

En este capítulo veremos los resultados obtenidos, el juegacooooo.

**Capítulo 6: Conclusiones y propuestas**

En este capítulo lo vamos a petar.

**Bibliografía**

**Anexo I.** Manual de usuario

## OBJETIVOS DEL PROYECTO

En este capítulo hablaremos de los objetivos que se pretenden conseguir en el desarrollo de este proyecto fin de carrera.

### Objetivo principal

El objetivo principal de este PFC es la aplicación del paradigma de desarrollo basado en línea de productos software que, además es un paradigma de desarrollo que empieza a implantarse en muchas empresas. Para ello, se desarrollará una línea de productos software sobre juegos de mesa debido a que comparten un amplio conjunto de características comunes (existencia de tablero, gestión del turno, posibilidad de jugar con dados, etc.).

### Objetivo secundario

El objetivo secundario de este proyecto es la implementación de una aplicación cliente-servidor, como resultado de uno de los posibles productos de la línea de productos software diseñada, que permita a los usuarios jugar a distintos juegos de mesa, gestionando la creación de partidas de diferentes tipos de juegos, la posibilidad estar unido a varias partidas a la vez, permitir tanto el juego online como el juego contra jugadores controlados por el ordenador, etc.

### Herramientas y medios empleados

A continuación se indicarán todas las herramientas y medios utilizados durante el desarrollo de este proyecto.

#### Middleware de comunicaciones

Un middleware es un software de conectividad (basado en el modelo cliente-servidor) que aporta abstracción sobre la complejidad y heterogeneidad de las comunicaciones de un sistema dado (3). Es decir, a través de la definición de unas interfaces de comunicación entre los distintos sistemas que forman una aplicación distribuida se eliminan los problemas de comunicación entre ellos pudiendo tener ambos diferentes arquitecturas hardware, sistemas operativos, protocolos de red o lenguajes de programación. Además dichas características dan al sistema una escalabilidad mucho mayor, lo cual es uno de los objetivos de este proyecto.

Por tanto se ha elegido en este caso middleware **ZeroC Ice 3.4.2** (4) que cumple con todas las características comentadas anteriormente.

#### Lenguaje de programación

Se ha elegido como lenguaje de programación **Java** (5)tanto para el cliente como para el servidor debido a que se consigue la independencia de la arquitectura y del sistema operativo de la máquina donde se despliegue. La versión del kit de desarrollo de Java (**JDK**) utilizado ha sido la **1.7**.

Para el desarrollo de la interfaz de comunicaciones entre cliente y servidor se ha utilizado el lenguaje **Slice** (6) que proporciona el middleware de comunicaciones ZeroC Ice y que permite realizar la definición de los tipos y las interfaces de los objetos utilizados por una aplicación. Esta descripción es independiente del lenguaje de implementación por lo que no importa el lenguaje en el que estén escritos tanto el cliente como el servidor.

#### Entorno de desarrollo

El entorno de desarrollo elegido ha sido **Eclipse 4.2 Juno** (7) por ser una herramienta de desarrollo “open source”, multiplataforma y que ofrece gran cantidad de plugins para la mayoría de tecnologías que puedan necesitarse en el desarrollo de aplicaciones. En este caso se han utilizado los siguientes plugins:

* **Windows Builder Pro** (8) es un plugin que facilita el diseño y creación de las interfaces en java mediante la tecnología Swing.
* **Slice2Java** es un plugin aportado por ZeroC Ice que genera automáticamente a partir de la definición de la interfaz de comunicaciones y objetos comunes entre cliente y servidor mediante el lenguaje Slice, todas las clases necesarias para el funcionamiento de la aplicación.

#### Base de datos

Para la implementación de la base de datos se ha elegido la utilización del sistema gestor de base de datos relacional **MySQL Server 5.5** (9) y su herramienta de gestión y diseño **MySQL Workbench 5.2 CE**.

Además para facilitar las transacciones con la base de datos se ha decidido la utilización del framework de persistencia para Java llamado **Hibernate** (10)que permite manejar de forma transparente dichas transacciones.

#### Herramienta de análisis y diseño

Debido a las características de este proyecto fin de carrera, se necesita una herramienta con potencia suficiente para desarrollar de forma adecuada la línea de productos software. En este caso se ha elegido **Visual Paradigm** ya que permite el uso del lenguaje de modelado **UML,** así como la creación de perfiles necesarios para el desarrollo de este proyecto.

#### Herramientas de gestión

Para la gestión del proyecto se ha decidido la utilización de la herramienta **JIRA** (11)de la compañía Atlassian, que es una aplicación basada en web para el seguimiento de errores, incidentes y gestión operativa de proyectos. Debido a la magnitud del proyecto y los recursos humanos del mismo, esta herramienta permite organizar de manera ágil las diferentes tareas del proyecto, así como llevar un control de las mismas y el tiempo dedicado a la consecución de cada una de ellas.

También se ha utilizado un control de versiones mediante el uso de un repositorio **Subversion** alojado en **Google Code** y gestionado mediante la utilización de la herramienta **Tortoise SVN 1.7.7.**

## ESTADO DE LA CUESTIÓN

### Introducción

Como se ha comentado anteriormente, las Líneas de Productos son una estrategia muy utilizada en numerosas ramas de la industria que consiste en la producción de múltiples productos utilizando una infraestructura común. Este enfoque se está introduciendo también en la ingeniería del software, en la medida en que ésta va siendo cada vez más una actividad industrial que necesita mejorar su eficiencia y optimizar sus costes.

El interés en las líneas de productos software surge desde el estudio de la reutilización cuando los desarrolladores se dan cuenta de que pueden incrementar los beneficios de la reutilización aprovechando arquitecturas software completas en lugar de reutilizar únicamente componentes software individuales.

La idea sobre una línea de productos no es nueva. Hay ejemplos de líneas de productos en la historia antigua (véase Figura 3.1), las pirámides de Egipto pueden haber sido la primera línea de productos. Como ejemplo más moderno de una línea de productos puede considerarse la industria aeronáutica con la creación de los aviones Airbus A-318, A-319, A-320 y A-321, que comparten características comunes como pueden ser los motores, el sistema de navegación o el equipo de comunicación.

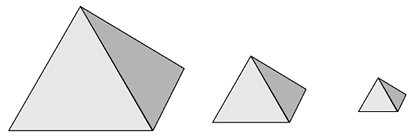


Figura 3.1 Ejemplo de línea de productos en la historia antigua

El objetivo en el modo tradicional de desarrollo de software es desarrollar un único sistema, es decir, desarrollar un único sistema de forma independiente. En las líneas de productos software el alcance se ve ampliamente aumentado considerando una familia de sistemas software en lugar de un único sistema. Esta aproximación incluye analizar qué características o requisitos funcionales tienen en común todos los productos software, que características son opcionales y cuales son alternativas. Después del análisis de las características, el objetivo es diseñar la arquitectura de la línea de productos que tendrá **componentes comunes** (necesarios por todos los miembros de la línea), **componentes opcionales** (necesarios solo por algunos miembros de la línea) y **componentes variantes** (diferentes versiones según las necesidades de los diferentes miembros de la línea). Para modelar y diseñar familias de sistemas, es necesario extender los conceptos de análisis y diseño de un desarrollo tradicional para soportar las líneas de productos software.

#### Reutilización

La reutilización en el software ha sido el objetivo en Ingeniería del software desde 1968, cuando el término “ingeniería del software” fue concebido. La reutilización en el software tradicional se realiza mediante el desarrollo de librerías de componentes con código reutilizable. Esta aproximación requiere que se establezca una librería de componentes reutilizables y un sistema para indexar, localizar y distinguir entre componentes similares. Esta aproximación tiene varios problemas serios como puede ser el manejo de una gran cantidad de componentes o como se ha indicado antes, la necesidad de distinguir entre componentes que son parecidos pero no idénticos.

Los componentes reutilizables de una librería son bloques de construcción que se utilizan en el desarrollo de un nuevo sistema. Dichos componentes son considerados atómicos e idealmente invariables cuando son reutilizados, aunque alguna pequeña adaptación pueda ser necesaria. Dependiendo del modelo de desarrollo, la librería puede contener componentes funcionales u orientados a objetos.

Aparte de ciertos dominios específicos, como las librerías matemáticas, los beneficios de la reutilización en el desarrollo de software tradicional son bastantes limitados en general, la reutilización es relativamente baja y se centra sobre todo en la reutilización de código.

En lugar de reutilizar componentes individuales, es mucho más beneficioso reutilizar un diseño entero o un subsistema, es decir, reutilizar los componentes y las conexiones entre cada uno de ellos. La reutilización de una arquitectura tiene mucho más potencial que la reutilización de componentes debido a su mayor granularidad y a que está centrada en la reutilización de requisitos y diseños.

### Líneas de productos software

La aproximación más prometedora para la reutilización de una arquitectura software es desarrollar una línea de productos software. Una línea de productos software captura explícitamente las características comunes y la variabilidad entre los sistemas que forman la línea de productos.

**Una línea de productos software (*LPS*)** se define por tanto como “*un conjunto de sistemas software, que comparten un conjunto común de características (features), las cuales satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base (core assets) de una manera preestablecida*” (2).

De acuerdo con esta definición, vale la pena considerar el desarrollo de una línea de productos cuando se gana más analizando los sistemas de forma colectiva que individualmente, es decir, cuando los sistemas tienen más características en común de las que los hacen diferentes entre ellos.

Los “activos base” se pueden definir como todo aquello que constituye la base de la línea de productos (12). Por lo tanto se puede considerar como “activos base” la arquitectura, los componentes reutilizables, los modelos de dominio, los requisitos, la documentación, la planificación, los presupuestos, los plantes de prueba, los casos de pruebas, plantes de trabajo, etc. El principal activo base es la arquitectura y al conjunto de todos ellos se le llama plataforma.

No todos los desarrollos de software pueden llegar a ser Líneas de Producto. La condición básica es que exista un conjunto de sistemas con características comunes. En función de las características del dominio y del nivel de riesgo e inversión que se quiera asumir, dichos productos software constituirán una línea de productos o no.

Algunos enfoques proporcionan una **arquitectura genérica** para la línea de productos donde se representa las características comunes de la línea pero ignora toda la variabilidad. Cada aplicación empieza con una arquitectura general para más tarde adaptarla manualmente según las necesidades. Aunque este enfoque proporciona un mejor punto de partida en comparación con el desarrollo de sistemas sin reutilización, falla en la captura de todo el conocimiento sobre la variabilidad de los diferentes productos de la línea.

Una aproximación más deseable es modelar explícitamente tanto las características comunes como la variabilidad de la línea. Dependiendo del enfoque de desarrollo (funcional u orientado a objetos), las características comunes de la línea se describe en términos de módulos comunes, clases o componentes y la variabilidad se describe en términos de módulos, clases o componentes variables.

El modelado de las características comunes y la variabilidad es una de las actividades más importantes en el desarrollo de las líneas de productos software. Para ello se introduce el concepto de **puntos de variación** como una forma de modelar la variabilidad en los casos de uso y UML.

#### Modelado de la variabilidad

La variabilidad en una línea de productos software puede modelarse a nivel de requisitos software o a nivel de diseño. El enfoque más extendido para modelar la variabilidad a nivel de requisitos software es a través del modelado de características (Feature modeling).

##### Modelado de características

Las **características** (features) son un concepto importante en las líneas de producto software porque representan los requisitos o características reutilizables de una línea de productos. El concepto de característica aplica a todas las líneas de producto y no solo a las líneas de producto software. Si se considera una línea de productos sobre vehículos, muchos modelos diferentes de coches pueden compartir características comunes. Por ejemplo, una línea de productos software de vehículos podría tener un chasis común (que representa una característica común), una elección sobre el tamaño del motor (donde cada motor representa una característica alternativa) y un navegador opcional (que representa una característica opcional). Para cada coche individual, que en este caso sería uno de los productos resultantes de la línea, un comprador no tendría opción a elegir chasis debido a que es una característica común, podría elegir entre cada uno de los motores disponibles y también elegir si tener o no un sistema de navegación.

En cuanto a las líneas de productos software, el método **FODA (feature-oriented domain analysis)** es un método que utiliza las características organizadas en un árbol de características. Las características deben ser obligatorias, opcionales o mutuamente excluyentes. El árbol de características es una composición jerárquica de características en la que algunas ramas son obligatorias, otras opcionales y otras mutuamente excluyentes. En este método las características deben ser funcionales (hardware o software), características no funcionales (seguridad, rendimiento), o parámetros (rojo, verde, azul).

El concepto de característica no es un concepto orientado a objetos y por lo tanto no es usado en el modelado UML para sistemas independientes. Más tarde se describirá como representar las diferentes características mediante UML.

##### Modelado en diseño

Existen diferentes técnicas para modelar la variabilidad a la hora del diseño entre las que se incluyen la parametrización, ocultar información y la utilización de la herencia.

1. **Parametrización**

Los parámetros pueden ser utilizados para introducir la variabilidad en las líneas de productos software. El valor de los parámetros definido en los componentes de la línea de productos es lo que otorga la variabilidad. Diferentes miembros de la línea de productos tendrán diferentes valores asignados a dichos parámetros.

Este enfoque utiliza cuatro tipos de parámetros:

* **Parámetros de compilación.** Los valores son asignados en tiempo de complicación. Esta aproximación a veces es utilizada para compilar código condicionalmente dependiendo del valores asignado al parámetro.
* **Parámetros de configuración.** Los valores son asignados en la configuración del sistema.
* **Parámetros de inicio de ejecución.** Los valores son asignados en inicialización del sistema.
* **Parámetros dirigidos por tablas.** Algunas aplicaciones reutilizables se configuran con parámetros que están almacenados en tablas. Este puede ser una buena forma de permitir a los usuarios configurar la aplicación

Esta aproximación puede tener el problema de manejar un gran número de parámetros sin tener la certeza de poder asegurar que unos parámetros no entran en conflicto con otros.

1. **Ocultación de información**

En este enfoque, diferentes versiones de un componente tienen la misma interfaz pero diferente implementación para diferentes miembros de la línea. La variabilidad se oculta dentro de cada versión del componente. En este caso las variantes son las diferentes versiones del mismo componente que deben tener una interfaz común. Esta aproximación funciona bien si los cambios se limitan a componentes individuales y no a cambios en la interfaz de los componentes. Desde el punto de vista de la reutilización, el ingeniero selecciona un componente dentro de un conjunto limitado y lo utiliza en la aplicación.

La mayoría de los métodos de líneas de productos utilizan este enfoque.

1. **Herencia**

Utilizando la herencia, diferentes versiones de una clase utilizan la herencia para heredar operaciones de la superclase para después redefinir las operaciones de la interfaz de la superclase o extender dicha interfaz añadiendo nuevas operaciones.

Con el extenso uso de las técnicas orientadas a objetos, la variabilidad utilizando herencia se ha vuelto muy común en las líneas de producto software.

En muchas líneas de productos se necesita una combinación de estos tres enfoques. El desarrollo orientado a objetos ayuda a este cometido ya que es capaz de soportar estos enfoques para modelar la variabilidad.

#### Introducción al desarrollo basado en UML

El campo de la reutilización de software ha evolucionado desde la reutilización de componentes individuales hacia la reutilización a gran escala con las líneas de productos software. Los enfoques de modelado de software moderno, como UML, proporcionan una mayor profundización en el entendimiento y gestión de las características comunes y la variabilidad modelando las líneas de producto desde diferentes perspectivas.

La metodología de desarrollo de software basado en UML utilizado en este proyecto, es un método para desarrollo de líneas de productos software llamada **PLUS** (Product Line UML-Based Software Engineering). La metodología PLUS extiende las metodologías de modelado basadas en UML que son utilizadas para desarrollar sistemas individuales para abordar las líneas de productos software. Con PLUS, el objetivo es modelar explícitamente las características comunes y las variabilidades en una línea de productos software. PLUS proporciona un conjunto de conceptos y técnicas para extender las metodologías de diseño basadas en UML y los procesos de desarrollo de sistemas individuales para gestionar las líneas de productos software.

Para entender la línea de productos y desarrollar un modelo para ella, un analista necesita considerar muchas diferentes perspectivas de la línea. Un modelo de una línea de productos es por lo tanto una representación desde múltiples puntos de vista de la línea donde cada punto de vista presenta una perspectiva diferente. Los diferentes puntos de vista son desarrollados de forma iterativa. Analizando los diferentes puntos de vista, un analista puede tener un mejor entendimiento de la línea.

### Ingeniería de líneas de productos software

A continuación se presenta la metodología PLUS desde la perspectiva del proceso de desarrollo de software. PLUS es una metodología compatible con el Proceso Unificado de Desarrollo de Software y el proceso de modelo en espiral y orientada al desarrollo de líneas de productos software.



#### Modelo de proceso evolutivo

El modelo de proceso evolutivo es un modelo de proceso que elimina la distinción tradicional entre el desarrollo del software y su mantenimiento. En lugar, los sistemas evolucionan a lo largo de varias iteraciones. Por lo tanto, los sistemas desarrollados con este enfoque necesitar ser capaces de adaptarse a cambios en los requisitos durante cada iteración. Además, debido a que los nuevos sistemas software a menudo se derivan a partir de sistemas ya existentes, este modelo de proceso toma la perspectiva de una línea de productos software permitiendo su desarrollo.

El modelo de proceso evolutivo se divide en dos procesos o ciclos de vida (ver Figura 3.2):

1. **Ingeniería de dominio**. Durante este proceso, se analizan los elementos comunes y la variabilidad de la línea de productos en función de los requisitos de toda la línea. Esta actividad consiste en desarrollar un diagrama de casos de uso de la línea, un modelo de análisis de la línea, la arquitectura de la línea y los componentes reutilizables. Los artefactos producidos durante este proceso son almacenados en un repositorio de la línea de productos software.
2. **Ingeniería de producto.** Durante este proceso, se desarrolla un producto concreto de la línea de productos software. En lugar de empezar desde cero, como normalmente se hace con los sistemas individuales, los desarrolladores hacen uso de todos los artefactos desarrollados durante el proceso de Ingeniería de dominio. Dado todos los requisitos del producto concreto, el modelo de casos de usos, modelo de análisis y la arquitectura de la línea de productos se adapta para derivar los modelos y la arquitectura correspondiente al producto concreto. En este momento se utilizan los componentes necesarios del repositorio de la línea de productos software para desplegar el producto concreto.

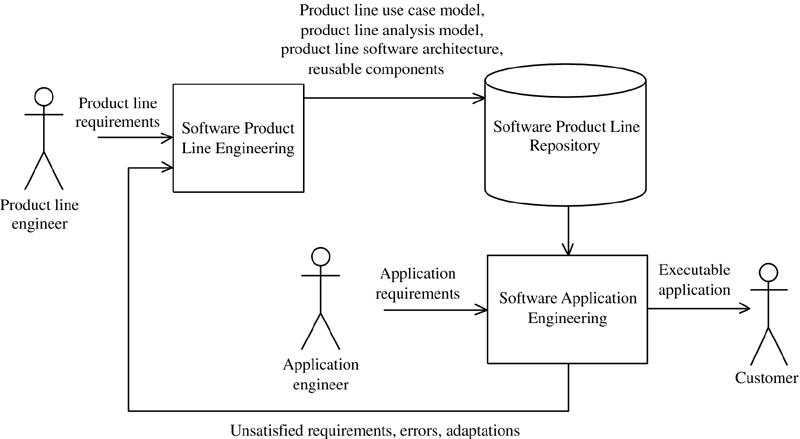
****

Figura 3.2 Modelo de proceso evolutivo

#### Fases del proceso de Ingeniería en líneas de producto software

El modelo de proceso evolutivo es un proceso de desarrollo de software altamente iterativo dirigido por el concepto de casos de uso. Durante la ingeniería de requisitos, los requisitos funcionales de la línea de productos se definen en términos de actores y casos de uso. Durante el análisis, cada casa de uso de la línea de productos se realiza para describir los objetos que participan en el caso de uso y sus interacciones. Durante el diseño, se desarrolla la arquitectura de la línea basada en componentes software. Todos los artefactos producidos durante cada una de estas etapas se almacenarán en el repositorio de la línea de productos software (Ver Figura 3.3). A continuación se describirán más en profundidad cada una de estas fases.

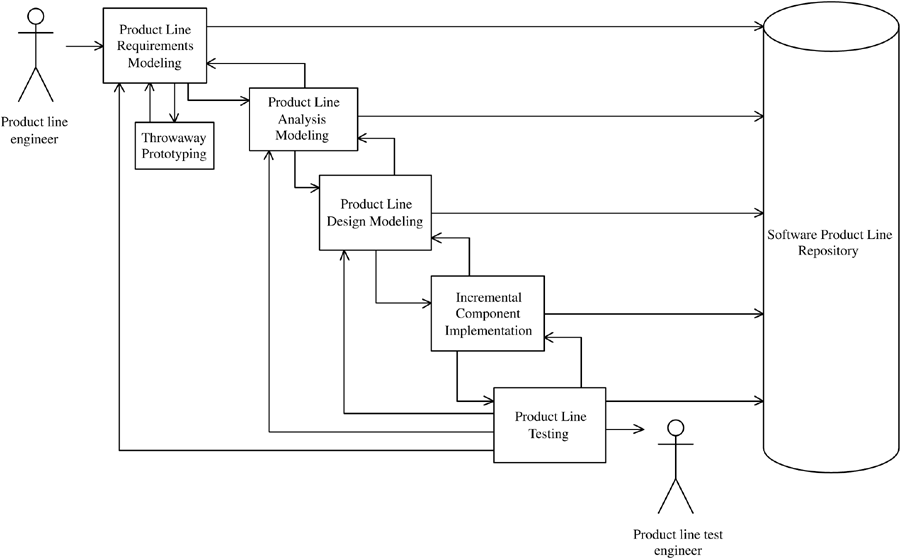


Figura 3.3 Ingeniería de la línea de productos software.

##### Ingeniería de requisitos

Durante la fase de ingeniería de requisitos, se desarrolla un modelo de requisitos que consiste en un modelo de casos de uso y un modelo de características. Igual que en el desarrollo de sistemas individuales, el modelo de casos de uso define los requisitos funcionales de la línea de productos software en términos de actores y casos de uso. Sin embargo, el modelo de casos de uso para una línea de productos necesita ser extendido para modelar los elementos comunes y la variabilidad en la línea de productos a través del desarrollo de casos de uso **kernel**, **opcionales** y **alternativos**. También hace una descripción narrativa para cada caso de uso. Si los requisitos de la aplicación no son bien entendidos, se puede desarrollar un pequeño prototipo que ayude a clarificar los requisitos.

Para las líneas de productos software, se necesita también el desarrollo de un **modelo de características**. Una **característica** (feature) es un requisito que proporciona uno o más miembros de la línea. En particular, son usadas para diferenciar entre los productos de la línea y por tanto determina la funcionalidad común y variable. Las características pueden ser funcionales o parametrizadas.

##### Análisis

Durante la fase de análisis, se desarrollan un modelo estático y dinámico. El *modelo estático* define las relaciones estructurales entre las clases del dominio del problema. Las clases y sus relaciones se describen en diagramas de clases. Después se desarrolla un *modelo dinámico*en el que los casos de uso procedentes de la ingeniería de requisitos se realizan para mostrar los objetos que participan en cada caso de uso y cómo interactúan. Los objetos y la interacción entre ellos se describen en diagramas de interacción, normalmente serán diagramas de comunicación o diagramas de secuencia.

Para las líneas de productos software, el *modelo estático* hace referencia a los elementos comunes y variabilidades entre los miembros de la línea de productos categorizando las clases entre kernel, opcionales y variantes. El *modelo dinámico* hace referencia a los elementos comunes y la variabilidad desarrollando diagramas de interacción para cada caso de uso. La aproximación comienza con el desarrollo de los casos de uso kernel y después evolucionar ese modelo para que incluya los casos de uso opcionales y alternativos.

##### Diseño

Durante la fase de diseño, se diseña la arquitectura basada en componentes de la línea, es decir, el modelo de análisis se mapea a un entorno operacional. El modelo de análisis (que se centra en el dominio del problema) se mapea al modelo de diseño (que se centra en el dominio de la solución). El diseño de la arquitectura empieza considerando patrones arquitectónico que incluyen tanto la estructura de la arquitectura (las dependencias entre componentes) y la dinámica de la arquitectura (en términos de cómo los componentes se comunican unos con otros).

##### Implementación incremental de componentes

Después de diseñar la arquitectura, el siguiente es paso es la implementación de manera incremental de los componentes. En esta fase, se selecciona un subconjunto de la línea para ser implementado en cada incremento. Los desarrolladores de la línea de productos determinan un subconjunto eligiendo los casos de uso que se incluirán en ese incremento, así como los componentes que participen en esos casos de uso. La implementación incremental comienza con los casos de uso kernel, seguido de los casos de uso opcionales y alternativos, de acuerdo con la secuencia establecida en el *modelo dinámico.*

Esta fase de implementación consiste en la realización de un diseño detallado, codificación y pruebas unitarias de los componentes del subconjunto elegido.

##### Pruebas en líneas de productos

Las pruebas en las líneas de productos incluyen tanto pruebas de integración como pruebas funcionales de la línea.

Durante las pruebas de integración, se prueban los componentes de cada iteración. Las pruebas de integración para el incremento se basan en los casos de uso seleccionados en dicho incremento y se desarrollan para cada uno de los casos de uso seleccionados. Estas pruebas es una forma de pruebas de caja blanca en las que se prueban las interfaces entre los componentes que participan en cada caso de uso.

Las pruebas funcionales prueban el sistema en referencia a los requisitos funcionales. Estas pruebas es una forma de pruebas de caja negra basadas en los casos de uso.

Después de que las pruebas para el incremento actual sean satisfactorias se procede a la construcción de un nuevo incremento. Sin embargo, si se detectan problemas significativos en el incremento del software desarrollado puede ser necesaria una nueva iteración sobre el mismo incremento.

En el enfoque de las líneas de productos software, las pruebas de integración y funcionales pueden llevarse a cabo para los casos de uso kernel. También se pueden realizar algunas pruebas con los componentes opcionales y alternativos, aunque es más normal que se aplace hasta el proceso de Ingeniería de producto.

#### Enfoques de construcción de una línea de productos software

Existen principalmente dos estrategias a la hora de desarrollar una línea de productos software: el **enfoque sustractivo** y el **enfoque aditivo.**

##### Enfoque sustractivo

El enfoque sustractivo es más aplicable cuando ya existen los sistemas que se pretenden analizar y modelar. En primer lugar se analizan y documentan los casos de uso de cada sistema individual para luego integrarlos todos en el que será el modelo de casos de uso de la línea de productos software. Los casos de uso *kernel* formarán el *kernel* del dominio de la aplicación y los casos de uso utilizados solo por alguno de los sistemas individuales o un subconjunto de ellos, formarán los casos de uso *optional* y *alternative.*

En definitiva, todas las variabilidades se contemplan a nivel de ingeniería de dominio y a nivel de producto se eliminan aquellas que no corresponden al producto concreto que se está desarrollando.

##### Enfoque aditivo

El enfoque aditivo se centra en el desarrollo del kernel de la línea de productos software. Primeramente se analiza la funcionalidad de la línea de productos con el objetivo de determinar la funcionalidad común de la misma. Normalmente, este análisis comienza con el desarrollo de los casos de uso kernel, es decir, los casos de uso que son comunes a todos los miembros de la línea de productos.

Durante la evolución de la línea de productos, se incorporarán nuevos casos de uso opcionales y alternativos que se irán determinando en cada una de las iteraciones. En resumen, a nivel de ingeniería de dominio se trabaja únicamente con las *common features* para luego a nivel de ingeniería de producto incorporar las variabilidades que correspondan.

Este enfoque tiene la ventaja de que la evolución se incorpora en el propio proceso de desarrollo por lo que más tarde la línea puede seguir evolucionando. El enfoque aditivo funciona mejor cuando se comienza a desarrollar un nuevo producto de la línea y la funcionalidad *kernel* se puede determinar antes que la funcionalidad variable.

.

## MÉTODO DE TRABAJO

Para el desarrollo de la línea de productos software sobre la que se centra este proyecto se elegido utilizar una metodología comentada anteriormente llamada PLUS. Esta metodología está basada en el Proceso Unificado de Desarrollo (PUD)

### PLUS: Ingeniería de Software para Líneas de Productos basadas en UML

PLUS (Product Line UML-Based Software Engineering) es una metodología para el desarrollo de LPS propuesta por Gomaa que se apoya en UML y que se basa en el Proceso Unificado de Desarrollo de software. A continuación se describirá como se integra PLUS con el Proceso Unificado de Desarrollo Software y las fases de las que consta.

#### Proceso Unificado de Desarrollo Software

El Proceso Unificado de Desarrollo tiene tres características esenciales: está dirigido por casos de uso, está centrado en la arquitectura y es iterativo e incremental.

**Proceso dirigido por Casos de Uso**

Los casos de uso son una técnica de captura de requisitos que fuera a pensar en términos de importancia para el usuario, y no sólo en términos de funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema. Además de ser una herramienta para especificar los requisitos del sistema, los casos de uso también permiten guiar el diseño, la implementación y las pruebas. Los Casos de Uso constituyen un elemento integrador y una guía de trabajo.

**Proceso centrado en la arquitectura**

La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo.

El desarrollo centrado en la arquitectura:

* Involucra los aspectos estáticos y dinámicos más significativos del sistema
* Está relacionado con la toma de decisiones que indican cómo tiene que ser construido el sistema
* Ayuda a determinar en qué orden deben irse obteniendo los artefactos relacionados

Además la definición de la arquitectura debe tomar en consideración elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución por lo que deber ser flexible durante todo el proceso de desarrollo. En el caso del Proceso Unificado de Desarrollo además de utilizas los Casos de Uso para guiar el proceso se presta especial atención al establecimiento en fases tempranas del desarrollo de una arquitectura que no se vea fuertemente impactada ante cambios posteriores.

En la figura 3.4 se ilustra la evolución de la arquitectura durante las fases del Proceso Unificado de Desarrollo. Se tiene una arquitectura más robusta en las fases finales del proyecto. En las fases iniciales lo que se hace es ir consolidando la arquitectura por medio de *baselines* y se va modificando dependiendo de las necesidades del proyecto.

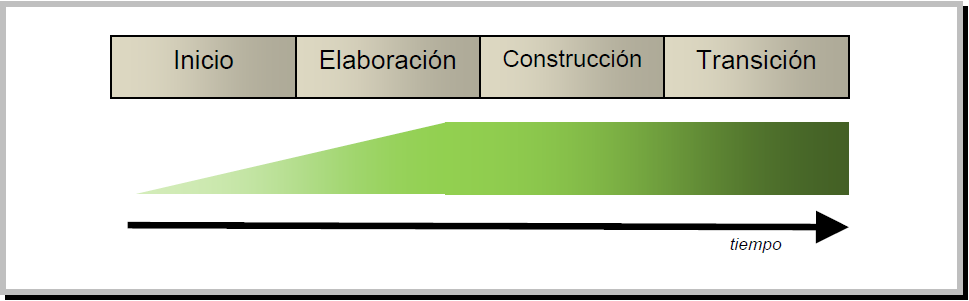


Figura 4.1 Evolución de la arquitectura del sistema

**Proceso iterativo e incremental**

El equilibrio correcto entre los casos de uso y la arquitectura es algo muy parecido al equilibrio de la forma y la función en el desarrollo del producto, lo cual se consigue con el tiempo. Para esto, la estrategia que se propone en el Proceso Unificado de Desarrollo es tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o mini proyectos. Así el equilibrio entre casos de uso y arquitectura se va logrando durante cada mini proyecto, y se va alcanzando a medida que se avanza en el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración (un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales), del cual se obtiene un incremento que produce un crecimiento en el producto.

El proceso iterativo e incremental consta de una secuencia de iteraciones. Cada iteración aborda una parte de la funcionalidad total, pasando por todos los flujos de trabajo relevantes y refinando la arquitectura. Cada iteración se analiza cuando termina. Se puede determinar si han aparecido nuevos requisitos o si han cambiado los existentes, afectando a las iteraciones siguientes. Durante la planificación de los detalles de la siguiente iteración, el equipo también examina cómo afectarán los riesgos que aún quedan al trabajo en curso. Toda la realimentación de la iteración pasada permite reajustar los objetivos (agenda y presupuesto) para las siguientes iteraciones. Se continúa con esta dinámica hasta que se haya finalizado por completo con la versión actual del producto.

Una iteración puede realizarse por medio de una cascada como se muestra en la figura 3.5. Se pasa por los flujos fundamentales (Requisitos, análisis, diseño, implementación y pruebas), también existe una planificación de la iteración, un análisis de la iteración y algunas actividades específicas de la iteración. Al finalizar se realiza una integración de los resultados con lo ya obtenido en las iteraciones anteriores.

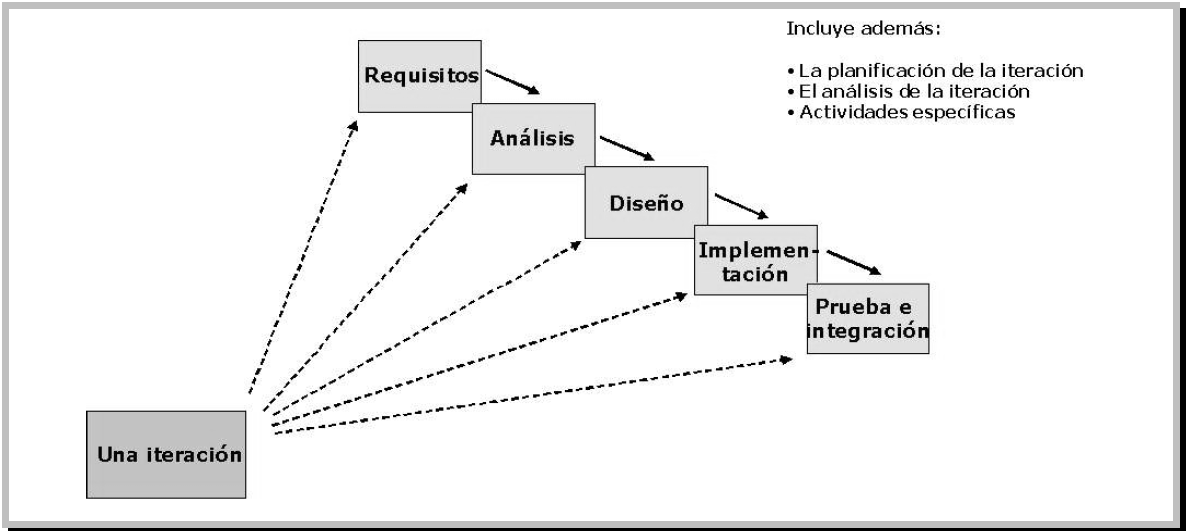


Figura 4.2 Iteración en el Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo divide el desarrollo en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menos hincapié en las distintas actividades. En la figura 3.6 se muestra cómo varía el esfuerzo asociado a los flujos de trabajo según la fase en la que se encuentre el proyecto.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una línea base de la arquitectura. Durante la fase de inicio, las iteraciones ponen mayor énfasis en las actividades de modelado de negocio y de requisitos. En la fase de elaboración, las iteraciones se orientan al desarrollo de la *baseline* d la arquitectura, abarcan más los flujos de trabajo de requisitos, modelo de negocio (refinamiento), análisis, diseño y una parte de la implementación orientado a la línea base de la arquitectura. En la fase de construcción se lleva a cabo la implementación del producto por medio de una serie de iteraciones. Por cada iteración se selecciona un conjunto de casos de uso en función de su prioridad, se refina su análisis y diseño y se procede a su implementación y prueba. Después se realizan tantas iteraciones como sean necesarias hasta que se termine la implementación de la nueva versión del producto. En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega el cliente.

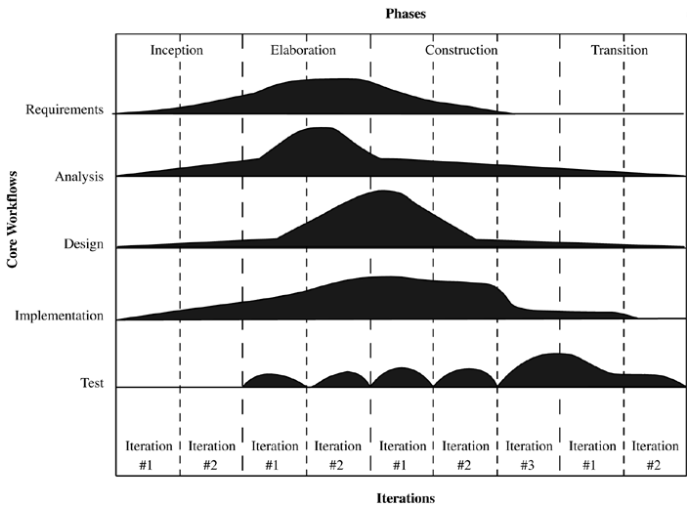


Figura 4.3 Esfuerzo en las actividades según la fase del proyecto

#### Integración de PLUS con el Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo proporciona un detalle considerable sobre los aspectos del ciclo de vida y algunos detalles sobre la metodología a utilizar. Como se ha visto anteriormente los flujos de trabajo del Proceso Unificado de Desarrollo son Requisitos, Análisis, Diseño, Implementación y Pruebas. En PLUS cada fase del ciclo de vida se corresponde con un flujo de trabajo del Proceso Unificado de Desarrollo. Las fases de PLUS tienen el mismo nombre que las flujos de trabajo del Proceso Unificado de Desarrollo.

##### Inicio

En esta primera iteración se realiza un estudio de viabilidad para determinar si la línea de productos es viable y cuál es el alcance (tamaño, funcionalidades, identificación de *commonalities* y *variabilities* y número estimado de productos).

Durante esta fase, se identifican los casos de uso iniciales para cada producto potencial de la línea, se crea un diagrama de clases de contexto inicial y una primera aproximación del *feature model* en el que se comienza a determinar aquellas *features* que son *kernel*, *optional* o *variant*.

##### Elaboración

Esta fase se dividirá en dos partes:

1. **Kernel primero (*Kernel first)***

En esta primera parte se revisa y desarrolla el modelo de casos de uso con más detalle. Se determinan los casos de uso *kernel*, *optional* y *variant* y se hace un análisis más en profundidad de las *features*. El *feature model* obtenido será ahora el principal elemento que guíe en la identificación de los requisitos comunes y variables. Los elementos del *kernel* se modelan mediante diagramas de análisis y diseño, construyéndose un esbozo inicial de la arquitectura del sistema.

1. **Evolución de la línea de productos (*Product line evolution)***

En esta segunda parte se planifica la evolución de la línea considerando los casos de uso *optional* y *variant*, las *features* y los puntos de variación. Se realizan diagramas de interacción para los casos de uso *optional* y *variant*, y máquinas de estado para las clases *optional*  y *variant*, que tienen una dependencia fuerte del estado. El diseño arquitectónico del sistema, esbozado en la iteración anterior, se extiende ahora para incluir componentes *optional* y *variant*.

##### Construcción

En las diferentes iteraciones de las que puede constar esta fase, se construyen los componentes que conforman el *kernel*. Para ello, se procede realizando el diseño detallado, la codificación y las pruebas unitarias de cada componente. También se hacen pruebas de integración de los componentes del *kernel*.

##### Transición

Al llegar a esta fase se dispone de una versión mínima ejecutable de la línea de productos software consistente en los componentes *kernel* y cualquier componente *default*. También se realizan pruebas funcionales y de integración de los componentes del *kernel*.

El sistema, en su estado actual, se puede entregar a los ingenieros de producto. Además el *kernel* también es un miembro de la línea de productos por lo que puede ser liberado a los usuarios para la realización de pruebas adicional y comenzar el uso del sistema.

##### Iteraciones adicionales

Puede haber iteraciones adicionales en las que se desarrollen componentes *optional* y *variant*. El jefe de proyecto debe decidir si estos componentes se implementan durante la ingeniería de dominio o, por el contrario, se dejan para más adelante, para el nivel de ingeniería de producto. Si se realizan iteraciones adicionales implicará la construcción y pruebas de la nueva arquitectura evolucionada con los nuevos componentes *optional* y *variant*.

#### Ingeniería de producto

Durante la Ingeniería de producto, la arquitectura de la línea de productos software se adapta y ajusta para obtener un producto de la línea. La obtención de un nuevo producto implica considerar todos los requisitos del producto individual seleccionando las *features* específicas del producto de la línea de productos. Dada la arquitectura del producto, se instancian, interconectan y despliegan los componentes apropiados de la línea de productos software.

Del mismo modo que el proceso de ingeniería de líneas de productos puede ser integrado con el Proceso Unificado de Desarrollo, también puede integrarse la Ingeniería de producto. En lugar de comenzar con el modelado de casos de usos, la Ingeniería de producto comienza con los artefactos de la línea de productos software almacenados en el repositorio de la línea.

##### Inicio

Durante la fase de inicio, se analizan y categorizan los requisitos del producto a desarrollar. El objetivo es determinar cuando el nuevo producto es un miembro viable de la línea de productos, es decir, cuando las *features* del nuevo producto coinciden con las de la línea de productos software. Si las *features*  no coinciden hay dos alternativas:

1. Hay suficiente diferencia entre las *features* como para que sea mejor el desarrollo individual del nuevo producto.
2. La línea de productos software puede evolucionarse añadiendo los requisitos que falten mediante una evolución incremental.

##### Elaboración

Esta fase se dividirá en dos partes:

1. **Evolución**

Si es necesario evolucionar la línea de productos, esta fase es similar a la fase de elaboración de la Ingeniería de Dominio en la que el modelo de la línea de productos software se evoluciona considerando las nuevas características optional y alternative. Si no es necesario evolucionar la línea, esta fase puede ser ignorada.

1. **Adaptación**

En esta parte, las *features* del nuevo producto guían la adaptación de la arquitectura de la línea de productos. Basándose en estas *features*, el modelo de casos de uso, el modelo de análisis y el modelo de diseño se ajustan para encajar con las necesidades del nuevo producto.

1. **Construcción**

En esta fase se seleccionan los componentes *kernel*  del repositorio de la línea de productos, los componentes *optional* y *variant* y se implementa cualquier componente del que previamente se hayan especificado sus interfaces pero que todavía no hayan sido implementados. La implementación de los nuevos componentes incluye el diseño detallado, la codificación y las pruebas unitarias de los componentes. También se realizan pruebas de integración de los nuevos componentes con los existentes.

1. **Transición**

Durante esta fase se realizan pruebas intensivas sobre el nuevo producto desarrollado. Se desarrollan casos de prueba en base a los requisitos funcionales especificados en los casos de uso seleccionados para el nuevo producto.

## RESULTADOS

### Ingeniería de dominio

#### Inicio

### Ingeniería de producto

## CONCLUSIONES Y PROPUESTAS

## BIBLIOGRAFÍA

1. **Diez, Oscar y Trujillo, Salva.** *Fábricas de Software: experiencias, tecnologías y organización.* s.l. : Ra-Ma, 2010.

2. **Clements, Paul C. y Northrop, Linda M.** *Salion, Inc.: A Software.* 2002.

3. **Colouris, G., Dollimore, J. y Kindberg, K.** *Distributed System: Concepts and Design.* s.l. : Addison Wesley, 2005.

4. **Documentación ZeroC Ice.** [En línea] http://doc.zeroc.com/display/Ice/Ice+Manual.

5. **Página web de Java.** [En línea] http://www.oracle.com/technetwork/java/index.html.

6. **Documentación sobre el lenguaje Slice.** [En línea] http://doc.zeroc.com/display/Ice/The+Slice+Language.

7. **Página web de Eclipse.** [En línea] http://www.eclipse.org.

8. **Página web de Windows Builder Pro.** [En línea] https://developers.google.com/java-dev-tools/wbpro/.

9. **Página web de MySQL.** [En línea] http://dev.mysql.com/doc/.

10. **Página web de Hibernate.** [En línea] http://hibernate.org/docs.

11. **Pagina web de Atlassian JIRA.** [En línea] http://www.atlassian.com/software/jira/overview.

12. **Software Engineering Institute.** *A Framework for Software Product Line Practice.* [En línea] 2001. http://www.sei.cmu.edu/plp/framework.html.