GUÍA METODOLÓGICA PARA LA CONSTRUCCIÓN DE FÁBRICAS DE SOFTWARE

ANDRÉS FELIPE ALZATE ÁLVAREZ
CAROLINA MONTOYA ÁLVAREZ
ARBEY DARÍO MUÑOZ GÓMEZ

ASESOR: ALEJANDRO GUTIERREZ SEGURO

DEPARTAMENTO DE INFORMÁTICA Y SISTEMÁS
ESCUELA DE INGENIERÍA
UNIVERSIDAD EAFIT
MEDELLÍN
2008

AGRADECIMIENTOS

A nuestros padres y amigos que nos apoyaron en todo momento y tuvieron la paciencia de soportar las incomodidades que les pudimos haber generado.

A nuestros proveedores de información y experiencia Jubel Alberto Correa Barco, César Andrés Ahmed, Juan Camilo Montoya y Pedro Luis Velázquez quienes sin interés de recibir algo a cambio nos apoyaron en el desarrollo del proyecto.

Especiales agradecimientos a nuestro asesor, mentor y amigo Alejandro Gutiérrez Seguro quien representó para nosotros un gran apoyo y una fuente inagotable de conocimiento convirtiéndose en un integrante más de nuestro equipo de trabajo.

9. GLOSARIO

A

- *Activo de software*:
  Es un elemento o materia prima que sirve para el desarrollo de un producto software.

- *Aplicación Software*:
  Programa informático que permite a un usuario usar la computadora para realizar tareas específicas.

- *Artefactos*:
  Es un producto tangible resultante del proceso de desarrollo de software. Está mayormente asociado a métodos o procesos de desarrollo específicos, como el proceso unificado.

B

- *Base de datos*:
  Es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

- *Bugs:*
  Errores de Software que son el resultado de un fallo o deficiencia durante el proceso de creación de programas de ordenador o computadora (software).

C

- *Caché*:
  Es un conjunto de datos duplicados de otros originales, con la propiedad de que los datos originales son costosos de acceder, normalmente en tiempo, respecto a la copia en el caché.

- *Calidad de software*:

  Es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades y expectativas del cliente o usuario.

- *Casos de Prueba*:

  Son un conjunto de condiciones o variables bajo las cuales el analista determinara si el requisito de una aplicación software es parcial o completamente satisfactoria.

- *Ciclo de vida del software*:

  Describe el desarrollo de software, desde su fase inicial hasta su fase final. Define las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación.

- *Código fuente*:

  Es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar un programa. Es donde se encuentra descrito el funcionamiento del sistema.

D

- *Diseño de Software*:

  Se define como el proceso de aplicar ciertas técnicas y principios con el propósito de definir un dispositivo, un proceso o un sistema con suficientes detalles como para permitir su interpretación o realización física.

- *Documentación de usuario*:

  Es la descripción de cómo el usuario debe usar la aplicación software.

- *Depurar*:

  Consiste en la revisión de la aplicación generada con el fin de eliminar los posibles errores que puedan existir en esta. También persigue la optimización del programa para que su funcionalidad y velocidad sean las máximas.

E

- *Etapa de Análisis*:
  Un conjunto de hechos, principios y reglas clasificadas y dispuestas de manera ordenada mostrando un plan lógico en la unión de las partes.

- *Estandarización*:
  Es la redacción y aprobación de normas que se establecen para garantizar el acoplamiento de elementos construidos independientemente, así como garantizar el repuesto en caso de ser necesario, garantizar la calidad de los elementos fábricados y la seguridad de funcionamiento.

G

- *General Packet Radio Service GPRS*:
  Es un servicio de datos móvil orientado a paquetes.

I

- *IDE*:
  Integrated Development Environment – (Entorno integrado de desarrollo). Aplicación compuesta por un conjunto de herramientas útiles para un programador.

- *Ingeniería de software*:
  Es la disciplina dentro del área de la informática que se encarga de la creación de software controlado y de calidad.

- *Institucionalizar los procesos*:
  Es el conjunto de principios reguladores que organizan la mayoría de las actividades de las empresas en pautas organizativas definidas, desde el punto de vista de los procesos que tengan definidos.

- *Interfaz de usuario*:
  Es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

- *Integración de productos*:

  Es el proceso mediante el cual todas las aplicaciones de una empresa se comunican entre sí, mediante procesos transparentes y en tiempo real

L

- *Log*:

  Es un registro de eventos que ocurre para un tema y periodo de tiempo en particular.

M

- *Mantenimiento del software*:

  Es una de las actividades más comunes en la Ingeniería de Software y es el proceso de mejora y optimización del software desplegado (es decir; revisión del programa), así como también corrección de los defectos.

- *Metadata o Metadato*:

  Se refiere a la descripción que se asigna a un objeto de aprendizaje, que sirve para catalogarlo y para solicitar su distribución.

- *Modelo de desarrollo de software*:

  Se compone de una mezcla de varios elementos. La filosofía detrás del desarrollo de software tiene amplia influencia en los otros dos elementos.

P

- *Producto*:

  Es un objeto que puede ser ofrecido a un mercado y pueda satisfacer un deseo o una necesidad.

- *Prototipo*:

  Un prototipo es un modelo a escala de lo real, no tan funcional como para que equivalga a un producto final, ya que no lleva a cabo la totalidad de las funciones necesarias del sistema final que sirve para proporcionar una retroalimentación temprana por parte de los usuarios acerca del sistema.

R

- *Requerimientos*:
  Es una solicitud que se hace legalmente para que se puedan satisfacer unas necesidades por medio de un sistema software.

- *Requisitos*:
  Es lo que un sistema software debe hacer para satisfacer las necesidades de un cliente.

S

- *Script*:
  Es un guión o conjunto de instrucciones que permiten la automatización de tareas creando pequeñas utilidades.

- *Servicio Web (Web Service)*:
  Es una funcionalidad que se expone vía Web, la cual debe responder a los protocolos de Internet estándares para funciones de accesibilidad, encapsulación de servicios y detección.

- *SOA (Service oriented architecture)*:
  Es una metodología para el desarrollo e integración de sistemas donde la funcionalidad es agrupada por procesos de negocio y encapsulada en servicios interoperables.

- *SQL*:
  Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

U

- *UML (Lenguaje Unificado de Modelado)*:
  Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

X

- *XML (Extensible Marhup Language)*:
  Es un lenguaje extensible de etiquitas desarrollado por el W3C, que permite definir la gramática de lenguajes específicos, en otras palabras, es una manera de definir lenguajes para diferentes necesidades.

- *XMI*:
  Es una especificación para el Intercambio de Diagramas. Es el nombre que recibe el estándar para el intercambio de meta-modelos usando XML.

W

- *Wifi*:
  Es un sistema de envío de datos sobre redes computacionales que utiliza ondas de radio en lugar de cables.

INDICE GENERAL

INDICE DE CONTENIDO.....2
INDICE DE ILUSTRACIONES.....5
INDICE DE TABLAS 6

INDICE DE CONTENIDO

AGRADECIMIENTOS **7**

INTRODUCCIÓN **8**

JUSTIFICACIÓN **9**

1. OBJETIVOS **10**

1.1. OBJETIVO GENERAL. 10

1.2. OBJETIVOS ESPECÍFICOS 10

2. ALCANCE **11**

3. ANTECEDENTES **12**

EVOLUCIÓN DEL DESARROLLO DE SOFTWARE 12

3.1. REUTILIZACIÓN 13

3.2. DESARROLLO DE SOFTWARE BASADO EN COMPONENTES 15

3.3. HERRAMIENTAS CASE 17

3.3.1. Composición de las Herramientas Case 17

3.3.2. Importancia de las Herramientas Case 17

3.3.3. Desventajas 18

3.4. DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS 19

3.5. LINEAS DE PRODUCTOS 21

3.5.1. Componentes básicos de la línea de producto: 23

4. MARCO TEÓRICO **24**

4.1. CONCEPTOS RELEVANTES 24

4.1.1. Patrones de software 24

4.1.2. Frameworks 31

4.1.3. Arquitectura de Referencia 37

4.2. DIFERENCIAS ENTRE FÁBRICAS DE SOFTWARE 46

4.3. DEFINICION DE FÁBRICA DE SOFTWARE 47

4.4. ETAPAS DEL PROCESO DE GENERACIÓN EN LAS FÁBRICAS DE SOFTWARE 53
4.4.1. Composición 53
4.4.2. Generación 54
4.5. DIFERENCIAS ENTRE MDA Y FÁBRICAS DE SOFTWARE 57
4.6. EJEMPLOS DE FÁBRICAS DE SOFTWARE 59

5. GUIA METODOLÓGICA 61
EJEMPLO PRÁCTICO, APLICANDO LOS PASOS PROPUESTOS EN LA GUÍA ..76

6. PROCESO DE DESARROLLO SOFTWARE DESPUÉS DE LA IMPLANTACIÓN DE LA FÁBRICA DE SOFTWARE 83
6.1. DEFINICIÓN DEL PROBLEMA 84
6.2. ANÁLISIS DEL PROBLEMA 84
6.3. DISEÑO DE LA SOLUCIÓN 85
6.4. GENERACIÓN DE LA APLICACIÓN 85
6.5. INTEGRACIÓN CON EDITOR DE CÓDIGO UTILIZADO 85
6.6. CONSTRUCCIÓN DE LA FUNCIONALIDAD ESPECÍFICA 86
6.7. PRUEBAS 86
6.8. MANTENIMIENTO 86

7. CONCLUSIONES 89

APENDICE A 92

APENDICE B 93

8. GLOSARIO 96

9. BIBLIOGRAFÍA 102

INDICE DE ILUSTRACIONES

Ilustración 1: Visión global de DSDM (Markus Völter y Jorn Bettin).....20
Ilustración 2: Conceptos básicos de la línea de producto por Charles W. Krueger.....22
Ilustración 3: Patrón de Arquitectura Software MVC (Modelo – Vista - Controlador) ...30
Ilustración 4: Diagrama de un Framework.....32
Ilustración 5: Estructura de Struts por Malcolm Davis 36
Ilustración 6: Diagrama de Hibernate. 36
Ilustración 7: Elementos de la arquitectura de referencia 38
Ilustración 8: Arquitectura de Referencia SOA. 44
Ilustración 9: Fábrica de Software definida por Jack Greenfield. 47
Ilustración 10: Componentes de la Fábrica de Software.....49
Ilustración 11: Esquema de Fábrica de Software de Jack Greenfield. 50
Ilustración 12: Esquema General de la generación de código de una Fábrica de Software.....53
Ilustración 13: Etapa de composición del proceso de generación de una Fábrica de Software.....54
Ilustración 14: Etapa de Generación del proceso de generación de una Fábrica de Software.....56
Ilustración 15: Esquema de trabajo de Web Client Software Factory de Microsoft. 60
Ilustración 16: Análisis del dominio del problema 64
Ilustración 17: Ejemplo del diagrama de Procesos.....77
Ilustración 18: Ejemplo de arquitectura de referencia. 80
Ilustración 19: Ciclo de vida implementado la fábrica de software.....83
Ilustración 20: Esquema de generación de código de Acceleo.....94

INDICE DE TABLAS

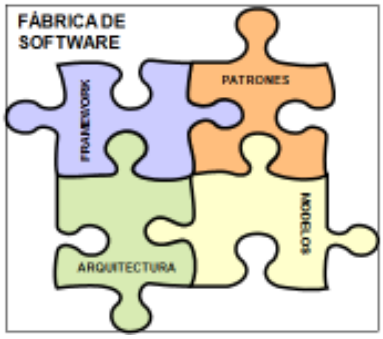
Tabla 1: Plantilla para la definición de un patrón propuesta por Gamma. 30
Tabla 2: Resumen de los pasos propuestos en la guía. 62

DEFINICIÓN FÁBRICA DE SOFTWARE

Una fábrica de software, es una aplicación software funcional, la cual por medio del uso de modelos de datos, diseños de aplicaciones y en general cualquier tipo de metadata, utilizando una serie de plantillas que definen la estructura o arquitectura que se desea tener en las aplicaciones, genera una aplicación o mínimamente la estructura de esta, facilitando la tarea de los diseñadores, desarrolladores, testers, y todos los implicados en el desarrollo de aplicaciones.

Los componentes fundamentales de las fábricas de software son los patrones, frameworks, arquitectura de referencia, mejores prácticas, modelos que abarcan todas las diferentes etapas por las cuales un software debe pasar para la automatización de los procesos y que hacen posible crear un esquema sobre el cual se podrá realizar el desarrollo y el mantenimiento de productos en particular.

Un ejemplo claro es un rompecabezas, todas las piezas deben estar en su lugar para poder formar la imagen, es lo mismo en la fábrica, todos sus componentes como lo son los patrones, frameworks, plantillas etc., deben funcionar de forma sincronizada de manera que cada una de las partes cumplan su función y que en conjunto se obtengan los resultados esperados.



ETAPAS DEL PROCESO DE GENERACIÓN EN LAS FÁBRICAS DE SOFWARE

En el proceso de generación de aplicaciones con la fábrica de software, se identifican dos etapas fundamentales para llevar a cabo este proceso, las cuales se describen a continuación:

Composición

En esta etapa, la fábrica de software define una serie de activos, los cuales componen una base de la aplicación a generar. Esta base en su mayoría cumple con las características no funcionales que debe tener la aplicación como son logger, seguridad, manejo de excepciones, entre otros.

Generación

Esta etapa es aquella en la cual la fábrica de software crea las aplicaciones generando automáticamente su código

fuente, utilizando como base una arquitectura de referencia y los insumos proporcionados por los diseñadores de la aplicación a generar, esto quiere decir que la fábrica tiene autonomía e inteligencia para decidir que archivos y funcionalidades debe crear de acuerdo a lo solicitado por el desarrollador.

GUÍA METODOLÓGICA PARA LA CONSTRUCCIÓN DE FÁBRICAS DE SOFTWARE

Paso 1: Modelado del Negocio

Recopila la información de todo el modelo de negocio haciendo que sea identificable las posibles necesidades que se puede suplir con aplicaciones software generadas por la fábrica.

Paso 2: Dominio del Problema

Analiza el modelo del negocio identificando y especificando cuales son los problemas que se presentan y se quieren solucionar con la creación de la fábrica de software.

Paso 3: Dominio de la Solución

Analiza los problemas identificados y define las posibles soluciones que se pueden emplear.

Paso 4: Diseño de la solución

Se seleccionan las soluciones óptimas para cada uno de los problemas identificados haciendo que sean posibles implementarlas genéricamente por la fábrica de software, mediante la definición de la arquitectura de referencia, los aspectos no funcionales y el esquema de la fábrica de software.

Paso 5: Seleccionar un generador de código

Se estudia y se selecciona en el mercado el generador de código que mejor se acople a las necesidades anteriormente identificadas

Paso 6: Creación de las plantillas de la fábrica

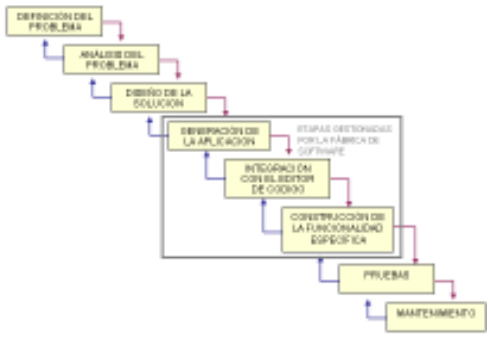
De acuerdo al generador de código seleccionado se genera las plantillas que describen la arquitectura de las aplicaciones que va a generar la fábrica.

Paso 7: Implementación de la línea de producción

Se construye el generador de aplicaciones, el cual maneja la lógica de creación de código fuente basándose en las plantillas y el generador de código seleccionado.

Paso 8: Pruebas de la fábrica de software

Como la fábrica es una aplicación software debe pasar por una etapa de pruebas que certifique su correcto funcionamiento.



Paso 9: Mantenimiento de la fábrica de software

La fábrica como todo software, debe permanecer en continuo crecimiento haciéndola cada vez más madura y confiable.

PROCESO DE DESARROLLO SOFTWARE DESPUÉS DE LA IMPLANTACIÓN DE LA FÁBRICA DE SOFTWARE

La introducción de una fábrica de software en un ambiente de desarrollo de software implica cambios grandes en la metodología que se utilizaba antes de esta, ya que aunque si bien la fábrica no genera la totalidad del código fuente, y la etapa de construcción no desaparece, si se modifican las actividades, tareas y tiempos de empleo para cada una de las etapas del ciclo de vida del proceso de desarrollo de software.

INTRODUCCIÓN

Hoy en día se ha detectado un crecimiento continuo a nivel del conocimiento y avance tecnológico en las empresas que basan sus procesos en aplicaciones software. Estos dos niveles son integrados por medio de una fábrica de software.
El conocimiento es representado en la concepción de la fábrica y en el dominio del problema que se quiere resolver. Por su parte, la tecnología es representada por las herramientas tecnológicas en las cuales se basa la fábrica de software y más aún en la ventaja que representa tenerla en cualquier entidad que construya aplicaciones software.

Para dar a entender que es una fábrica de software (Ver Sección 4.2), este documento hace referencia a algunos conceptos previos relevantes para la compresión de ésta, luego, presenta una definición de ella, en que consiste, para que sirve, cual es su estructura, entre otras características que hacen de este documento una herramienta útil para la construcción de una fábrica de software ya que no solo se enfoca en su definición sino en proponer una guía metodológica en la cual apoyarse al momento de su creación.

JUSTIFICACIÓN

El entorno económico actual donde se presenta un altísimo nivel de competencia, obliga a las empresas a buscar alternativas que les permitan ser más eficientes, eficaces y con gran cantidad de elementos que mejoren su rendimiento; dentro de esta búsqueda, muchas de ellas han encontrado su estrategia en apoyar en gran medida sus procesos de negocio en la tecnología para enfrentar estos retos.

Dentro de la dinámica anterior, es común que las organizaciones con el afán de responder a los constantes cambios que se presentan en su entorno, adquieran o desarrollen herramientas que apoyen sus procesos sin ser éstas las más adecuadas para acoplarse al negocio. Esto suele originar cuellos de botella difíciles de percibir o generan gran variedad de definiciones para una misma actividad.

Debido a esto, se evidencia la necesidad de crear metodologías y herramientas que ayuden a desarrollar aplicaciones software eficientes, con un alto grado de calidad y de una manera ágil, que cumplan con las exigencias y lineamientos de la compañía, buscando soluciones enfocadas en procesos y alineadas con los objetivos de las múltiples áreas del negocio.

1. OBJETIVOS

1.1. OBJETIVO GENERAL.

Proponer una guía metodológica para la creación de fábricas de software basadas en arquitecturas de referencia, patrones, frameworks y modelos que permitan el desarrollo de aplicaciones software que cumplan con la velocidad, flexibilidad, escalabilidad, control y calidad requeridas.

1.2. OBJETIVOS ESPECÍFICOS

- Presentar una serie de elementos que contribuyen al proceso de creación de fábricas de software.

- Resaltar la importancia del uso de fábricas de software como herramienta base para optimizar el proceso de desarrollo de aplicaciones software en los negocios.

- Enmarcar los conceptos que fundamentan las fábricas de software.

- Identificar los dos tipos de fábricas de software que existen en la actualidad tanto a nivel nacional como internacional.

2. ALCANCE

Este proyecto de grado propone una guía metodológica que sirve como base para la creación de fábricas de software.
Se explicará la importancia y necesidad de las fábricas de software en la actualidad, los dos tipos existentes, sus componentes principales como lo son patrones de software, arquitecturas y frameworks, explicando para cada uno de ellos los aspectos más relevantes que implican describir su uso, su importancia dentro de las fábricas de software y algunos ejemplos; además se revisarán algunas herramientas de apoyo para la creación de fábricas, diferencias entre fábrica como herramienta y como empresa, al igual que se definirán recomendaciones que ayudan al proceso de construcción de la fábrica.

Hay que tener en cuenta que este proyecto es netamente teórico, es por esto que no se desarrollará una implementación funcional para ninguno de los temas tratados.

3. ANTECEDENTES

EVOLUCIÓN DEL DESARROLLO DE SOFTWARE

El desarrollo de software ha pasado por varias fases, las cuales le han permitido madurar poco a poco hasta llegar a lo que se conoce ahora. Cuando el desarrollo de aplicaciones comenzó, no se pensaba que para llevar un problema a una solución sistemática era necesario realizar un análisis y un diseño, sino que se llegaba directamente a la implementación sin manejar una gestión detallada del mismo. Esto causaba en el trascurso del desarrollo una serie de inconvenientes que no permitía que las aplicaciones fueran flexibles en cuanto a su mantenimiento y/o adaptabilidad a diferentes ambientes.

Más adelante se vio la necesidad de aplicar prácticas exitosas y controladas que otras ciencias usaban y era posible adaptarlas a la ciencia computacional como las arquitecturas, los patrones entre otros. Estas técnicas de desarrollo adoptadas han permitido a través del tiempo que el desarrollo de aplicaciones software se haga de una manera gestionada, teniendo mayor control del mismo, que sea flexible en cuanto a cambios que se vayan a realizar, que la mantenibilidad del mismo sea menos compleja.

En la actualidad se desarrolla software a la medida o en palabras más comunes aplicaciones adecuadas a las necesidades de cada cliente, y el desarrollo de software genérico que son realizados para satisfacer las necesidades de un mercado “global”. Algunos términos comienzan a ser mencionados dentro del área de software como la familia de productos y los productos de población. Una familia de productos es un conjunto de aplicaciones con características comunes y pocas diferencias en cuanto a su construcción, diseño e implementación, es algo que se usa dentro de las aplicaciones; los productos de población son aquellos que son desarrollados con fines genéricos, pero a su vez pueden ser adaptados al negocio del cliente, otorgando funcionalidades específicas para él.

Estos dos conceptos hacen parte de lo que se denomina una línea de producto, la cual es una forma planeada y proactiva para reutilizar el software en un dominio de negocio.

A continuación se darán a conocer una serie de conceptos claves que preceden y fundamentan la llegada del concepto fábrica de software en la actualidad.

3.1. REUTILIZACIÓN

El concepto de reutilización de software, a través de los años ha pasado por diferentes etapas. La reutilización aparece como una alternativa para desarrollar aplicaciones software, permitiendo que el proceso de desarrollo sea eficiente, ágil y con un nivel de productividad más alto.

En el año de 1987 Freeman define la reutilización de la siguiente manera: "… Cualquier procedimiento que produce o ayuda a producir un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo mayor".

C. W. Krueger define la reutilización como "… un proceso de creación de sistemas software a partir de un software existente, en lugar de tener que rediseñarlo desde el principio".

Otra definición para la reutilización es la que expuso Sodhi & Sodhi en el año de 1999 "… la reutilización de software es el proceso de implementar o actualizar sistemas de software usando activos de software existentes".

Algunos de los beneficios que tiene la reutilización software son:

- Mejora de la productividad.
- Mejora de la calidad del software.
- Reducción del tiempo de desarrollo.
- Reducción de los costos.
- No tener que "reinventar las soluciones".
- Facilitar la integración de productos del ciclo de vida del software.

Como todo también presenta algunas desventajas como:

- Necesidad de invertir antes de obtener resultados.
- Necesidad de capacitación y formación del personal que hace parte de la empresa.
- Convencer al personal administrativo del uso de la reutilización.
- Incapacidad de institucionalizar los procesos de reutilización.

En la reutilización existen varios tipos y formas de realizarla, no solo se reutiliza código fuente, sino también otros elementos que comúnmente se ven en el desarrollo de una aplicación software. Estos elementos son:

- Planes de proyecto.
- Modelos de estimación de costos.
- Arquitectura.
- Especificaciones y modelos de requisitos.
- Diseños.
- Documentación de usuario y técnicas
- Interfaces hombre – máquina
- Datos
- Casos de prueba
- Patrones, esquemas, planillas
- Componentes de código

Una de las reutilizaciones de artefactos más conocidas en los últimos años es la reutilización de componentes software, el cual se logra crear, valga la redundancia, generando software a partir de componentes, permitiendo usar uno o varios de ellos en otros sistemas que sean similares, sin necesidad de volver a analizar y resolver problemas que de antemano ya están resueltas.

En la reutilización de software existen varias modalidades que actualmente son muy utilizadas en las compañías de desarrollo de software, estas modalidades son las siguientes:

- Individual: es cuando se utiliza algún activo de software que ya ha sido desarrollado en otros proyectos de la empresa y se añade al proyecto que se está realizando actualmente y para el cual es útil usarlo.
- Oportunista: es usar activos de software que han sido hechos por otras personas externas a la empresa que los comparten libremente, logrando así que problemas que se han presentado en diferentes tipos de proyectos y han sido solucionados, sean reutilizados en el proyecto en construcción siempre y cuando sean adaptables a este.
- Gestionada: es el tipo de reutilización en el cual se tiene un control sobre él con respecto a los cambios, es planificado, institucionalizado por la

empresa, en otras palabras, se crea una cultura en la empresa de reutilización de activos de software de los cuales se tiene un control periódico.

3.2. DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

El desarrollo de aplicaciones software basado en componentes está ligado a la reutilización, ya que este se basa en el uso de código elaborado previamente. Este desarrollo permite que al momento de hacer aplicaciones software no solo se piense en resolver un problema específico, sino también para su utilización en proyectos futuros, haciendo que los desarrollos disminuyan su complejidad y tiempo de implementación.

La definición de componente es "… una pieza de código preelaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar. Los componentes son los ingredientes de las aplicaciones, que se juntan y combinan para llevar a cabo una tarea". Otra definición que se puede tener de componente fue expuesta por Szyperski en el año de 1998, y lo definió así: "… Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio".
Luego de haber definido que es un componente, ahora se puede dar una definición de lo que es el Desarrollo de Software Basado en Componentes "… Es el paradigma de ensamblar componentes y escribir código para hacer que estos funcionen".

El desarrollo de software basado en componentes por sí solo no basta, necesita de algo más para que pueda ser adaptable a otras aplicaciones sin presentar problemas. Uno de los elementos clave para ello es establecer una arquitectura de software, la cual permite reestructurar de una manera organizada todo el desarrollo del componente. Las arquitecturas además de ordenar y definir una forma de hacer las cosas, permiten establecer como estos componentes se deben adaptar o comunicar con otros componentes que los vayan a usar, lo cual ayuda a que haya independencia entre ellos y flexibilidad a la hora de usarlos.

Algunas de las ventajas que existen de usar este paradigma, son:

- Reutilización del Software: ayuda a alcanzar un mayor nivel de reutilización software.
- Simplifica las pruebas: como lo que se va a reutilizar ya esta previamente probado, esto ayuda a que las pruebas sean más sencillas.
- Simplifica el mantenimiento del sistema: cuando existe un débil acoplamiento entre los componentes a usar, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario sin afectar otras partes del sistema.
- Mayor calidad: la calidad del desarrollo de la aplicación se va refinando o madurando con el paso del tiempo, ya que se va dando un proceso continúo de mejoramiento a los componentes usados en cada aplicación.
- Reduce los tiempos y costos de construcción aumentando el índice de productividad del proceso de desarrollo de software.

El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo en espiral (uno de los modelos del ciclo de vida existentes para el desarrollo de software). Este modelo incorpora los siguientes pasos para su desarrollo:

- Productos basados en componentes, son investigados y evaluados por el dominio de las aplicaciones en cuestión.
- Los problemas de integración son considerados en el desarrollo.
- Una arquitectura software es diseñada para acoplarse a los componentes.
- Los componentes son integrados dentro de la arquitectura.
- Unas pruebas exhaustivas son apropiadas para asegurar el buen funcionamiento.

3.3. HERRAMIENTAS CASE

Conjunto de herramientas, programas y ayudas que permiten que los analistas, técnicos, desarrolladores, y demás participantes implicados en la creación de aplicaciones, generen de manera automática programas que mejoren el ciclo de vida de desarrollo de software, es decir son diversas aplicaciones que ayudan a disminuir el desarrollo de software en relación al tiempo y el costo y a la automatización de éste dando como resultado una mejora en la calidad y en la productividad en el desarrollo de los sistemas.

3.3.1. Composición de las Herramientas Case

- **Repositorio o diccionario de datos,** en el cual se incluye toda la información que se va generando a lo largo del desarrollo del ciclo de vida.
- **Herramientas de Diseño,** son las herramientas de diagramación que permiten hacer modelos de lo que se pretende desarrollar.
- **Herramienta de prototipos,** herramientas que permiten modelar al usuario desde el momento mismo de la concepción una vista inicial de cómo sería el aspecto de la aplicación.
- **Generación de Código**, como su nombre lo dice son herramientas que permiten la generación de código a partir de los diagramas de diseños creados.
- **Generador de Documentación,** Este se basa en el repositorio para a partir de este, generar la documentación necesaria del desarrollo.

3.3.2. Importancia de las Herramientas Case

La importancia de las herramientas case radica en:

- Facilitan la realización de prototipos.
- Simplifican el mantenimiento de los programas.
- Permiten la reutilización de componentes de software.
- Automatizan el desarrollo de software, la documentación, generación de código entre otros.

- Aumentan la calidad del software
- Mejoran el tiempo y la definición aproximada de la planificación de un proyecto.

3.3.3. Desventajas

- El uso de métodos estructurados, debido a que no todas las organizaciones emplean este método
- Conflictos con el uso de diagramas, al igual que diagramas no utilizados.
- Falta de niveles estándar, es decir como no se tiene una herramienta estándar se debe elegir cuál de ellas es la que se soluciona el problema.
- Función limitada debido a que una herramienta está enfocada en la solución de un problema específico.
- Las herramientas CASE deben ser manejadas por personas con experiencia, en pocas palabras las tareas humanas siguen siendo críticas.
- En cuanto al mantenimiento de aplicaciones creadas con estas herramientas, se puede decir que éste es muy complicado, debido a que la estructura del código fuente y su nemotecnia no es muy entendible por lo cual se hace difícil encontrar errores o mejoras a éste.

3.4. DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS

En la actualidad se maneja una propuesta para el desarrollo de software llamada "Desarrollo de software dirigido por modelos" DSDM o MDD por sus siglas en ingles "Model Driven Development".

El desarrollo de software dirigido por modelos divide en dos partes el proceso de desarrollo de software. La primera es el dominio del problema, consiste en la capacidad de representar y describir la situación actual y la necesidad de la aplicación software. La segunda es el dominio de la solución, cuya función es convertir el dominio del problema e implementarlo en una tecnología en particular.

El DSDM es una estrategia de desarrollo de aplicaciones software, la cual se basa en la separación entre la especificación funcional del sistema y su implementación. Ésta separación es muy importante debido a que la especificación funcional es independiente de la plataforma o tecnología sobre la cual se basa la implementación, lo que lleva a proteger las inversiones en software ya que cuando sea el tiempo de iniciar la implementación, las tecnologías pueden haber avanzado y se puede elegir la mejor que se encuentre a disposición.
Adicionalmente se basa en el concepto de modelo, el cual es definido por Miller y Mukerji en 2003 como "… una descripción o especificación de un sistema y su entorno definida para cierto propósito. Un modelo es representado habitualmente como una combinación de elementos gráficos y de texto."
La idea de los modelos es abstraer tanto como sea posible el dominio del problema haciendo uso de leguajes de modelado como el UML. Estos lenguajes se basan en metamodelos, los cuales especifican el lenguaje, las relaciones y las reglas estructurales que hay entre los objetos del modelo diseñado.

Algunas de las razones por las que surgió esta propuesta son:

- El aumento de la complejidad de las aplicaciones software hace que tengan más interacciones con otras aplicaciones, más funcionalidades y necesidad de mayor rendimiento.
- La creciente evolución de las tecnologías hace que el desarrollo de software deba ser ágil para que se aproveche al máximo su vida útil.

- El 80% del costo de un producto software está destinado a su construcción.
- El diseño orientado a objetos.
- La definición de un lenguaje de modelado universal UML.

El siguiente mapa mental creado por Markus Völter y Jorn Bettin en mayo de 2004 facilita el entendimiento de todo lo que implica el desarrollo de software basado en modelos.

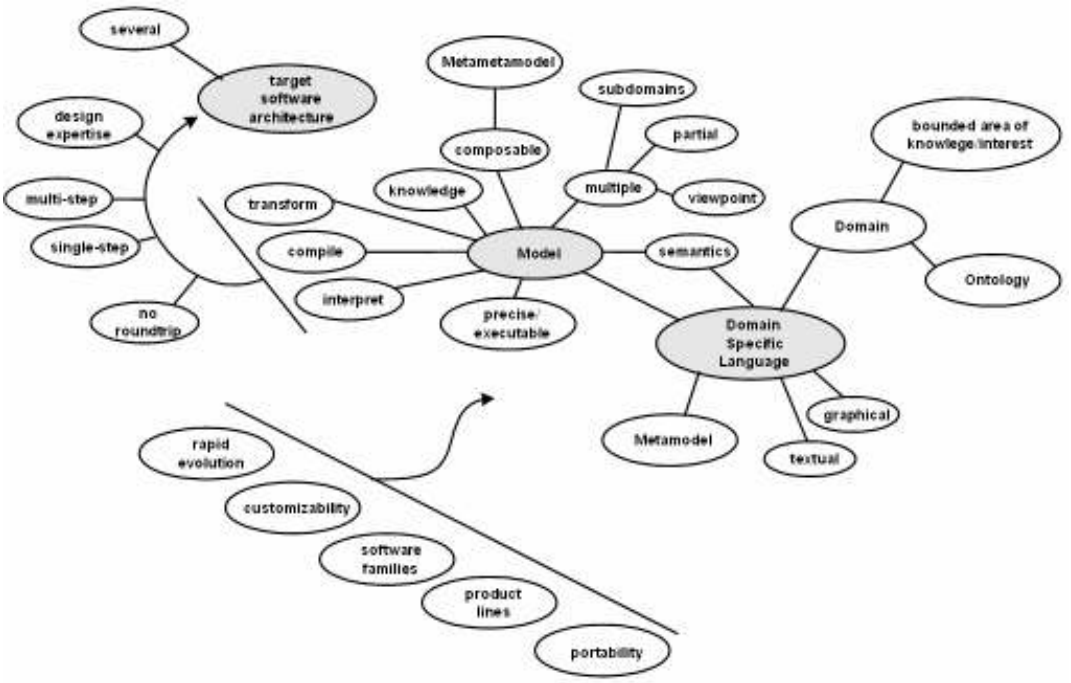


Ilustración 1: Visión global de DSDM (Markus Völter y Jorn Bettin).

3.5. LINEAS DE PRODUCTOS

Este tema es importante conocerlo de antemano para dar una mayor claridad al concepto y construcción de fábricas de software.

Se puede mencionar que una línea de producto es un recurso que utiliza la fábrica de software para su elaboración, ya que las fábricas construyen productos que pertenecen a un mismo dominio del problema.

Una línea de producto es un grupo de productos independientes entre sí, pero que unidos ofrecen un servicio único, estos productos están enfocados a usos similares, o poseen características similares. Este concepto está fundamentado en la reutilización, es decir, utiliza activos ya existentes para crear sistemas nuevos a partir de éstos.

Una de las definiciones más apropiadas para una línea de producto es aquella que define Clements en el año de 2001: "… se definen las líneas de producto de software como un conjunto de sistemas software, que comparten un conjunto común de características, las cuales satisfacen las necesidades especificas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base de una manera preestablecida."

En esta definición se menciona lo siguiente "… un conjunto de sistemas software…", a lo que se refiere este fragmento es que uno de los objetivos principales de una línea de productos software es el desarrollo de una familia o conjunto de productos y no uno solo en particular. Los productos de la familia que se generan a partir de la línea de producto son similares, ya que cumplen con las mismas características generales con las cuales fue construida la línea, las cuales hacen que estos sean considerados importantes por los clientes para describir, analizar y distinguir las necesidades que satisfacen los productos que son desarrollados a partir de la línea de productos software.

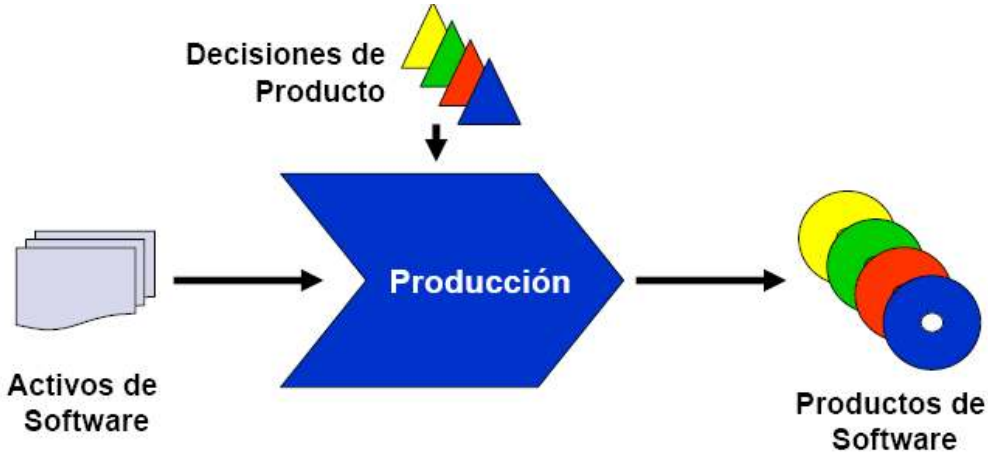


Ilustración 2: Conceptos básicos de la línea de producto por Charles W. Krueger

Una de las ventajas de utilizar una línea de productos software, es que ayuda a incrementar la productividad en el proceso de desarrollo de un aplicativo software, ya que los ingenieros de software reducen el esfuerzo aplicado al desarrollo y por lo tanto se reducen los costos del proceso de desarrollo. El uso de líneas de producto permite duplicar o triplicar la productividad con respecto a los modelos o enfoques tradicionales que existen para el desarrollo.

A diferencia de los enfoques tradicionales que se enfocan es en el producto y no en las similitudes que existen entre ellos, las líneas de productos software están pensadas para gestionar lo que es común a cada producto y en lo que es complementario a ellos como las variaciones que existen. En este caso la reutilización de componentes o código software ya no es oportunista como en los enfoques tradicionales, sino que se realiza una planificación sobre esta, y cada variación que se vaya incorporando al producto se realiza de manera sistemática y controlada, permitiendo una total gestión sobre cada producto generado por la línea.

La gestión sobre los componentes reutilizables que usa la línea de productos software permite que el mantenimiento de cada uno de los productos desarrollados por esta sean de fácil mantenimiento, reduciendo así los esfuerzos que se dedican a la parte de mantenimiento de software.

El desarrollo de software bajo el control de líneas de productos trae beneficios sobre la calidad que el producto puede tener al ser generado por esta.

La calidad de estos productos se puede medir desde dos puntos de vista importantes como lo son el cliente y la tasa de defectos del producto.

Con respecto al punto de vista del cliente, se puede decir que a mayor nivel de satisfacción que genere el producto al cumplimiento de las necesidades, mayor será el nivel de percepción de la calidad que tenga el cliente sobre el producto. Esto depende del diseño y el estudio que se haya realizado sobre el dominio del problema a la hora de crear la línea de producto software.
Con respecto a las tasas de defectos en los productos de la línea, éstos dependen de la reutilización de los elementos comunes, haciendo que la utilización continua de estos permita a través del tiempo estar demasiado depurados y probados, logrando que la tasa de defectos en los productos de la línea se vayan reduciendo.

Las líneas de productos software es una de las tantas búsquedas que se han hecho a través de la historia de la ingeniería de software para encontrar un equilibrio entre la calidad y los costos que conlleva el desarrollo de un aplicativo software.

3.5.1. Componentes básicos de la línea de producto:

Los componentes que se mencionan a continuación son los componentes básicos que debe tener una línea de productos

- Los Activos de software, son las entradas que se configuran para producir los productos.
- Modelo de decisiones, es el control que se aplica a las variables del sistema y que decisiones se deben tomar, al igual que se establecen los pasos para configurar los productos.
- Productos de Software, son las salidas de la línea de producto, es el conjunto de todos los productos.

4. MARCO TEÓRICO

4.1. CONCEPTOS RELEVANTES

A continuación se definirán algunos conceptos importantes para la correcta comprensión de las fábricas de software.

4.1.1. Patrones de software

Definición

Los patrones software son importantes en la definición y creación de una fábrica software, ya que es un activo que hace parte de ella, para definir la arquitectura que se va a usar y los frameworks que se adaptan a dicha arquitectura.

A continuación se presentan algunas de las definiciones que existen para los patrones software:

- Una definición de patrón software es la propuesta por Craig Larman "…El patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría indica la manera de utilizarlo en circunstancias diversas. Expresado en unas palabras más simples, el patrón es una descripción de un problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas".
- Según el grupo de los cuatro (Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides), "…Un patrón de diseño denomina, abstrae e identifica los aspectos claves de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases e instancias participantes, sus roles y colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se centra en un problema concreto, describiendo cuando aplicarlo y si tiene sentido hacerlo teniendo otras restricciones de diseño, así como las consecuencias y las ventajas e inconvenientes de su

> uso. Por otro lado, como normalmente tendremos que implementar nuestros diseños, un patrón también proporciona código de ejemplo en un lenguaje de programación, para ilustrar una implementación".

En conclusión, un patrón software es una solución dada a un problema que se ha presentado anteriormente, permitiendo diseñar un esquema genérico que pueda ser usado en diferentes ambientes software, sin necesidad de pensar en una solución al mismo problema.

Conceptos Claves

En los patrones de software es importante tener en cuenta algunos conceptos que son claves para la construcción y el uso de estos, como lo son la cohesión y el acoplamiento.

La cohesión es una condición que implica que todos los elementos de una clase trabajan en conjunto para lograr un fin. Es una medida relativa, en otras palabras depende de lo que cada persona piense que es la función para la cual fue creada la clase. Lo más importante es mantener una alta cohesión en el desarrollo de software.

El acoplamiento es la interdependencia que existe entre las clases. La interdependencia se ve cuando una clase para que pueda funcionar correctamente depende de la información de otras clases, con esto se rompe el encapsulamiento que es uno de los conceptos más importantes en la programación orientada a objetos. Para que esto sea efectivo a la hora de diseñar, y para el mantenimiento de los aplicativos, cuanto más bajo sea el acoplamiento mucho mejor.

Objetivos

Los patrones software deben cumplir algunos objetivos para que puedan cumplir la tarea para la cual están hechos, estos objetivos son los siguientes:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Como muchas personas piensan, los patrones de software no son una forma de imponer alternativas al diseño, sino que están hechos para resolver los problemas para los cuales han sido creados, lo cual no quiere decir que se deben forzar a que solucionen un problema especifico, deben ser usados de una manera adecuada para que puedan ser de ayuda en el desarrollo del software.

Tipos de Patrones y Aplicaciones

Los patrones de software se dividen en categorías, según el nivel de abstracción:

- *Patrones de Arqui*tectura: son aquellos patrones que expresan un esquema organizativo estructural fundamental para sistemas software.
- *Patrones de diseño*: aquellos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software.

- *Idiomas*: patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

Los patrones software además de que se dividen en categorías según su nivel de abstracción, se dividen por propósitos los cuales son:

- *Patrones Creacionales:* son aquellos patrones que se ocupan del proceso de creación de objetos. Algunos patrones creacionales son:

    - Abstract Factory o Fábrica Abstracta
    - Builder o Constructor Virtual
    - Factory Method o Método de Fabricación
    - Prototype o Prototipo
    - Singleton o Instancia Única

- *Patrones Estructurales:* son los encargados de tratar la composición de las clases y/o objetos. Algunos patrones estructurales son:

    - Adapter o Adaptador
    - Bridge o Puente
    - Composite u Objeto Compuesto
    - Decorator o Envoltorio
    - Facade o Fachada
    - Flyweight o Peso Ligero
    - Proxy

- *Patrones de Comportamiento:* son aquellos que se caracterizan por la forma en la cual las clases y los objetos interactúan y distribuyen responsabilidades. Algunos patrones de comportamiento son:

    - Chain of Responsability o Cadena de Responsabilidad
    - Command u Orden
    - Interpreter o Interprete
    - Iterator o Iterador
    - Mediator o Mediador
    - Memento o Recuerdo
    - Observer u Observador
    - State o Estado
    - Strategy o Estrategia
    - Tamplate Method o Método Plantilla
    - Visitor o Visitante

Las aplicaciones de algunos de estos patrones en ámbitos concretos ha permitido la creación de un lenguaje de patrones y libros completos de manos de diversos y reconocidos autores.
Entre los ámbitos en los cuales ha sido más notorio el uso de estos patrones son: en el manejo de interfaces de usuario, en la construcción de sistemas empresariales, en integración de sistemas y por ultimo en Workflow o Flujo de Trabajo.

Descripción de un Patrón

Para la creación y descripción de un patrón software no basta con solo hacer diagramas que expliquen o especifiquen el patrón, para esto existen una serie de plantillas ya definidas, que permiten explicar en

detalle un patrón. La plantilla que se presenta a continuación fue propuesta por Gamma:

Ítem	Descripción
Nombre del Patrón	Nombre corto y significativo, generalmente una o dos palabras
Clasificación	Usualmente para los patrones no existe ninguna clasificación, aunque por facilidad se agrupan en los diferentes tipos de patrones existentes
Propósito	Se menciona el problema de diseño que se quiere resolver y como el patrón que se define lo resuelve
También conocido como	Otros posibles nombres con los cuales se le pueden conocer al patrón
Motivación	Describe el escenario que ilustra el problema de diseño, y como las estructuras de clases y objetos del patrón resuelven el problema
Aplicabilidad	Describe en que situaciones se puede aplicar el patrón, ejemplos y formas de reconocer tales situaciones
Estructura	Se hace una representación gráfica del patrón que muestren sus elementos y relaciones constitutivas
Participantes	Especificación de las clases y objetos de las cuales está compuesto el patrón
Colaboraciones	Representación de cómo colaboran los participantes para cumplir con sus responsabilidades
Consecuencias	Especificación de las ventajas y desventajas que tiene el patrón a la hora de implementarlo
Implementación	Descripción de las dificultades, ,trucos o técnicas que se deberían tener presentes a la hora de usar el patrón
Código de Ejemplo	Especificación de código que ejemplifique como se debe usar el patrón
Usos Conocidos	Ejemplo donde haya sido usado el patrón
Patrones Relacionados	Especificación de los posibles patrones con los cuales está relacionado, las principales diferencias y

	los patrones con los cuales se debería usar

Tabla 1: Plantilla para la definición de un patrón propuesta por Gamma.

Ejemplo

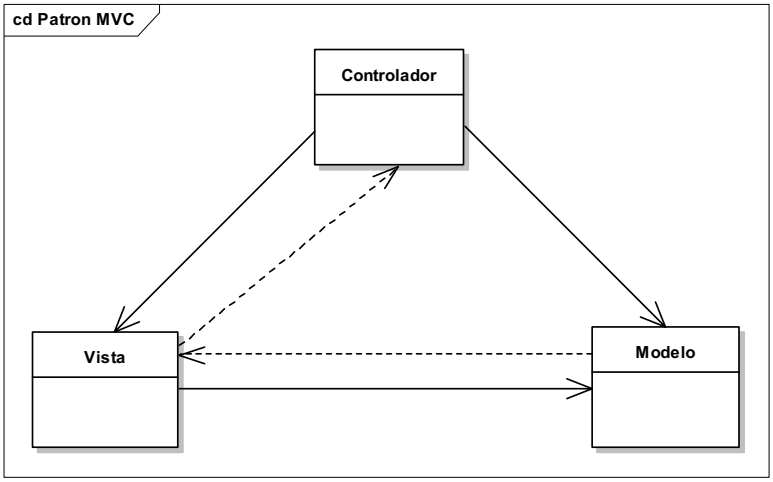


Ilustración 3: Patrón de Arquitectura Software MVC (Modelo – Vista - Controlador)

Este es el patrón arquitectónico MVC, como se puede ver en la figura 1 se pueden observar tres clases (Modelo – Vista - Controlador). La relación que no están punteada indican una asociación directa, y las líneas punteadas indican una asociación indirecta. Este es uno de los patrones más usados en el desarrollo de software, por esta razón la mayoría de los frameworks que existen hasta el momento soportan este tipo de arquitectura.

4.1.2. *Frameworks*

Un framework es una estructura o pieza de software, compuesta de un conjunto de componentes de diseño que facilitan la implementación de aplicaciones software mediante la reutilización y la definición de estándares de desarrollo para las diferentes necesidades. La idea de un framework es ayudar con las labores más engorrosas, menos atractivas y críticas, uniendo todo esto en una pieza de software que define un proceso fácil y único para la implementación de dicha labor.
Un ejemplo de esto es Hibernate, el cual es un framework Java para el manejo de la base de datos de una aplicación. Encapsula la conexión, las relaciones entre clases, la comunicación entre la base de datos y la aplicación, y la creación de scripts de base de datos ya sean inserciones, búsquedas, modificaciones entre otras.

En general, un framework es una aplicación software incompleta, la cual se debe completar con una serie de insumos que tenga definido dicho framework para que posteriormente se considere una aplicación completa. Lo anterior explica la importancia de los frameworks ya que con los insumos aportados, se diseña y personaliza la aplicación y lo único que hace el framework son las labores recurrentes de comunicación y transformación del proceso que se quiere aplicar.

Imaginemos un motor, este encapsula toda la lógica compleja de cómo hacer mover un objeto, pero el por sí solo no funciona, necesita insumos como la gasolina, el aire y el sistema de encendido y ahí si comienza su funcionamiento. De la misma manera se puede ver un framework, el por sí solo no hará lo que se necesita, pero si le entrega los insumos correspondientes, como un modelo de datos y una conexión a la base de datos, o una serie de vistas (JSP o html) y el modelo de clases el puede iniciar su funcionamiento.

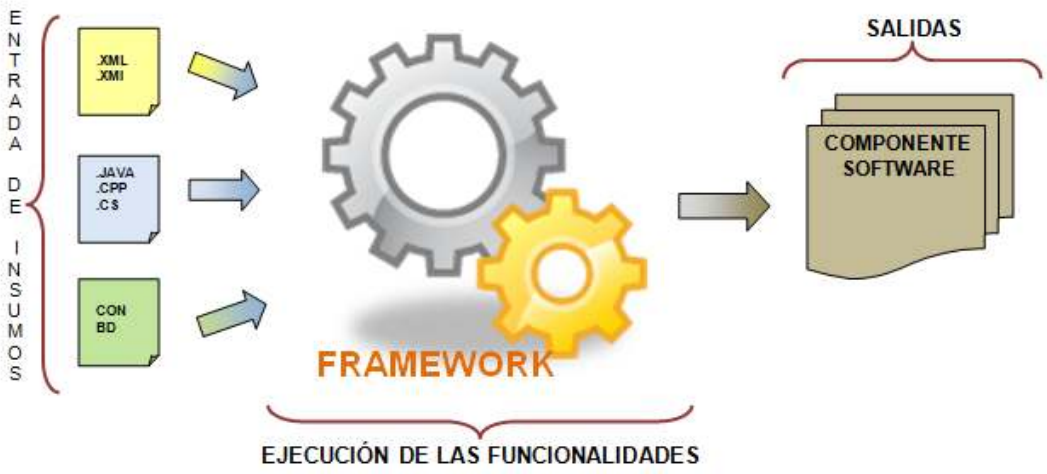


Ilustración 4: Diagrama de un Framework.

Objetivos principales de un framework

- Acelerar y agilizar todo el proceso de desarrollo de software.
- Reutilizar funcionalidades ya existentes.
- Hacer uso de las mejores prácticas de desarrollo.
- Permitir a los diseñadores y desarrolladores gastar más tiempo en las etapas de análisis y diseño del desarrollo del software y evitar el gasto innecesario de tiempo solucionando o desarrollando los requisitos de bajo nivel que todo sistema debe proveer.
- Hacer fácil el uso de tecnologías muy complejas.

Los frameworks pueden ser:

- La implementación de uno o varios patrones de diseño: ayudan a diseñar la aplicación, definiendo como puede ser la interacción entre los componentes, un ejemplo es el de diferenciar las capas de la aplicación como las capas del negocio y las de la presentación para esto se puede hacer uso de los tantos frameworks existentes que ayudan a implementar el patrón de desarrollo Modelo-Vista-Controlador.
- Pautas o procedimientos: esto implica que un framework define como hacer una tarea específica en la aplicación, por ejemplo la conexión a las bases de datos, para esto hay frameworks encargados de la interacción, entre la aplicación y la base de datos, facilitando la conexión y la obtención de datos.

Es muy importante no confundir un framework con una librería envolvente (Wrapper). La función de un framework no es proveer una serie de funcionalidades independientes, sino crear un proceso que hace uso de éstas para entregar una salida o un resultado deseado sobre un problema específico. Es decir, está diseñado y creado para un fin determinado por lo cual su uso es transparente para cualquier programador, en cambio las librerías envolventes son creadas para proveer una serie de funciones que son independientes entre sí, por lo que se debe conocer bien su funcionamiento para usarlo correctamente. En otras palabras, en una librería se deben llamar una serie de funcionalidades que se deben conocer para realizar una tarea específica, mientras que un framework las encapsula para que solo tenga que hacer uso de una sola función que realiza de forma ágil y eficiente el proceso que se quiere implementar.

Aunque es cierto que con el uso de librerías envolventes se puede realizar la misma funcionalidad que con un framework, es importante entonces recalcar las ventajas que tienen éstos sobre las librerías como lo son la flexibilidad y la extensibilidad. Un framework bien definido debe permitir extender sus funcionalidades haciendo que el desarrollador pueda implementarlas específicamente para el negocio, sin afectar el funcionamiento de éste. En cuanto a la flexibilidad un framework puede ser adaptado a cualquier tipo de negocio.

Tipos de frameworks

En la actualidad se pueden definir tres tipos de frameworks que son:

- Frameworks Web.
- Frameworks stand alone.
- Frameworks de propósito general.

Frameworks Web

Los frameworks Web son diseñados para facilitar la implementación de aplicaciones Web, entre ellos se encuentran frameworks orientados a la interfaz de usuario como Java Server Faces (JSF), frameworks orientados al control de eventos como STRUTS, orientados a la interacción entre las vistas y el servidor como AJAX (Asynchronous JavaScript and XML).

Frameworks Stand alone

Diseñados para las aplicaciones que no tienen la necesidad de estar expuestas en la Web, como las aplicaciones de escritorio. Entre estos encontramos frameworks orientados a la interfaz de usuario como Java SWING.

Frameworks de propósito general

Son diseñados para cualquier tipo de aplicación. Entre ellos encontramos frameworks para la interacción con la base de datos como HIBERNATE y DATABASEBEAN y frameworks de Logging como LOG4J.

Ventajas de utilizar frameworks

- Ayudan a no iniciar desde cero una aplicación.
- Reutilización de código existente.
- La mayoría de frameworks tienen soporte y suficiente documentación.
- Uso de las mejores prácticas.

- Sus funcionalidades son muy bien probadas por lo que el tiempo de pruebas en el desarrollo de la aplicación se reduce significativamente.
- Muchos frameworks son multilenguaje.
- Un framework bien concebido, es tolerante a fallas tanto propias como inducidas.

Desventajas de utilizar frameworks

- El uso de frameworks puede añadir código innecesario a las aplicaciones.
- Si no se tiene un conocimiento amplio acerca del framework, el tiempo que se pensaba ahorrar en su uso, puede ser malgastado en aprender a utilizar el framework.

Ejemplos de frameworks

STRUTS

Es un framework open – source, desarrollado en Java, el cual es una variante del patrón de desarrollo modelo vista controlador. Este framework tiene su propio controlador el cual permite la integración con otras tecnologías como las de acceso a datos, vistas, generación de documentos o trabajos con XML. STRUTS provee la comunicación entre el modelo y las vistas, haciendo que esto sea fácil y transparente para el desarrollador.

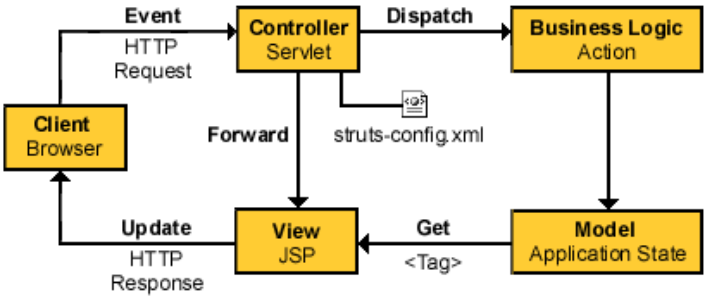


Ilustración 5: Estructura de Struts por Malcolm Davis

Un ejemplo clásico de la ventaja de utilizar el framework STRUTS, es el desarrollo de un sitio Web para un banco. En este caso, los analistas y desarrolladores pueden enfocarse en como hacer seguras las transacciones, retiros e ingresos, en el aspecto visual de la aplicación, y no tienen que preocuparse de la forma como controlaran la navegación y la interacción entre las paginas.

HIBERNATE

Hibérnate es un framework java que se encarga de la capa de persistencia de las aplicaciones. Provee la conexión a la base de datos, la ejecución de sentencias select, insert, update y delete mediante funciones java, permite la creación de querys dinámicos mediante la definición de su propio lenguaje llamado HQL y maneja el cache de los objetos persistentes. Todo lo anterior se realiza mediante la configuración de archivos XML para el mapeo de las tablas de la base de datos.

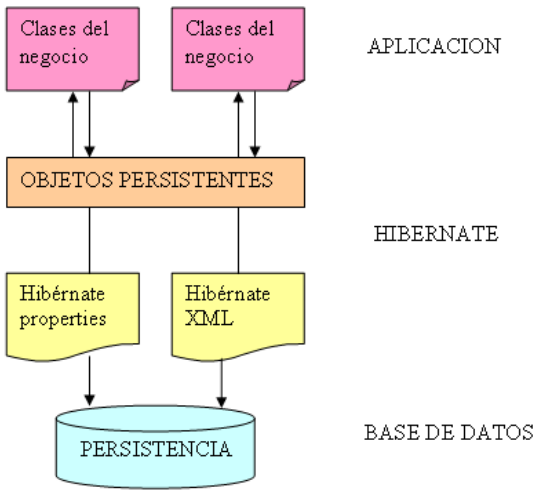


Ilustración 6: Diagrama de Hibernate.

4.1.3. *Arquitectura de Referencia*

Para comenzar a definir que es una arquitectura de referencia, inicialmente se debe dar a conocer que es arquitectura.

Arquitectura

Según la IEEE 1471 Arquitectura es:

+ "El nivel conceptual más alto de un sistema en su ambiente".
+ "Arquitectura es la organización fundamental de un sistema descrita en:
  - Sus componentes.
  - Relación entre ellos y con el ambiente.
  - Principios que guían su diseño y evolución."

Arquitectura de referencia:

Según IBM RUP una *arquitectura de referencia*

"… Es, en esencia, un patrón arquitectónico predefinidos, o conjunto de pautas, posiblemente parcial o totalmente instanciada, diseñado y probado para un uso en particular, las empresas y los contextos técnicos, junto con el apoyo artefactos para permitir su uso".

Partiendo de esto se puede definir la arquitectura de referencia como una arquitectura que contiene sus componentes y la relación entre estos con el medio ambiente pero en un dominio específico, siendo más claro es una arquitectura que encierra en ella los componentes comunes de los sistemas es decir, esta arquitectura permite que los proyectos de software que tengan características similares se desarrollen bajo los mismo principios previamente establecidos en ella.

La arquitectura de referencia describe la estructura lógica que será la base sobre la cual deben desarrollarse las aplicaciones, está materializada en plantillas y se genera a partir de patrones arquitectónicos, de un modelo de referencia y de la relación que existen entre ellos, además de esto contiene las características tanto funcionales como no funcionales básicas para el diseño de la arquitectura de software.

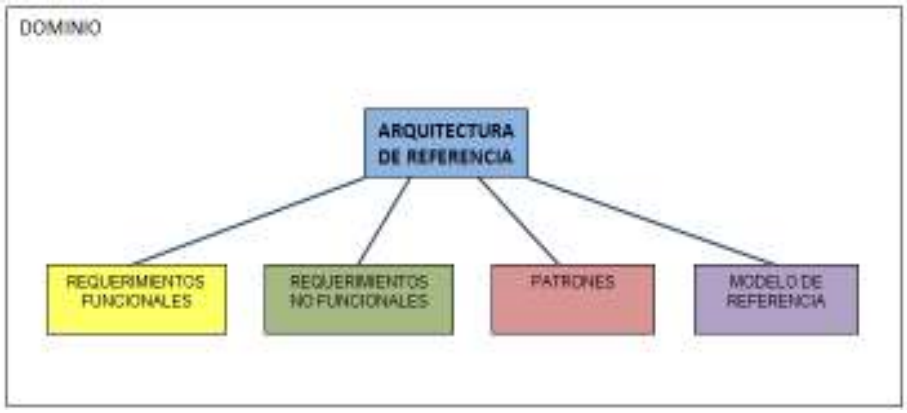


Ilustración 7: Elementos de la arquitectura de referencia

Los requerimientos no funcionales determinan como debe ser la operación de la arquitectura en su dominio, algunos de los requerimientos más conocidos son la portabilidad, costo y estabilidad entre otros.

Los requerimientos funcionales son los que definen el comportamiento del sistema internamente, es decir son los que debe cumplir el sistema especificados en los casos de uso.

Los patrones establecen subsistemas definidos, especifican las responsabilidades, las reglas y organiza las relaciones entre ellos.

Finalmente el modelo de referencia se utiliza para crear arquitectura de software, que influye fuertemente en la integridad y la estructura del software.

Conceptos Claves para la arquitectura de Referencia

- *Alta cohesión:*

  Este concepto hace referencia a la necesidad de encapsular en las clases de la aplicación única y exclusivamente funcionalidades que correspondan a esta, es decir que el comportamiento de dicha clase este enfocado en un único comportamiento. Las ventajas de aplicar la alta cohesión saltan a la vista, porque facilita la modificación no solo de una clase sino de un proceso de negocio, ya que la modificación de algún método afectara únicamente a los métodos que pertenecen a

dicha clase, haciendo que la labor de identificación, solución y prueba del cambio a nivel de toda la aplicación sea más fácil y rápida.

- *Bajo acoplamiento:*
  El acoplamiento es la relación que surge entre las diferentes clases de la aplicación, mientras estas relaciones sean mucho menores, las modificaciones sobre ellas serán más fáciles de implementar ya que habrá una mínima repercusión de dicho cambio sobre las otras clases que hagan referencia a este.

- *Mejores prácticas de desarrollo*:
  Se debe estudiar muy bien las mejores prácticas para el desarrollo de aplicaciones software, ya que estas pueden contribuir al éxito o fracaso de los proyectos. Estas prácticas, proponen metodologías de desarrollo, tecnologías a implementar, estándares y formas ágiles de construcción. Pero hay que tener en cuenta que el hecho de que sean denominadas como mejores practicas, no implica que funcionan en todos los ámbitos, por lo que su utilización debe surgir de un proceso de análisis de la mejor practica evaluada en el ámbito o entorno en el que se aplicará.

- *Patrones y Frameworks:*
  El uso de patrones y frameworks de software es una parte primordial de la definición de la arquitectura de referencia, ya que los patrones describen la mejor forma de hacer algo y los frameworks son cajas negras que tienen funcionalidades específicas muy bien implementadas y que mejoran y facilitan el desarrollo de las aplicaciones. Definir buenos patrones y frameworks no es tarea fácil, es necesario identificar cuales hay en el mercado, en el caso de los frameworks cuales son de dominio público y cuales necesitan licencia para su uso, se debe identificar la tecnología a utilizar, lenguaje de programación y necesidad a solucionar. En el apartado *conceptos relevantes* del marco teórico (punto 4 del documento), se explican más ampliamente los patrones, los frameworks y la arquitectura de referencia.

Características de lógica no funcional importantes:

- *Portabilidad:*

Esta característica define el grado de dependencia que tiene una aplicación software hacia el hardware u otro software sobre el cual puede correr. Para el caso de la arquitectura definida para la fábrica de software esto es crítico porque la fábrica debe ser capaz de funcionar casi bajo cualquier plataforma, ya que en entornos de desarrollo como el de las empresas desarrolladoras de software, hay gran variedad de sistemas operativos, lenguajes de programación y hardware entre las diferentes maquinas y proyectos que se abarcan, por esto la fábrica debe estar en la capacidad de ser útil para cualquiera de dichos escenarios. Por ejemplo con el uso de interfaces para la conexión a la base de datos, se puede lograr que la arquitectura de referencia provea el medio para ganar portabilidad en ese sentido, ya que para cada una de las bases de datos, se pueden definir diferentes clases que implementen dicha interfaz y cada una maneja una conexión a diferentes bases de datos.

- *Escalabilidad:*

  Una aplicación escalable es aquella que tiene la posibilidad de crecer en todos los sentidos, ya sea un crecimiento en cantidad de funcionalidades, crecimiento en la concurrencia de usuarios, crecimiento de datos entre otros. Esto se puede lograr de muchas maneras, por ejemplo definiendo o seleccionando frameworks o patrones de software que optimicen el rendimiento tanto a nivel de procesador, de memoria, de interacción con otras aplicaciones, etc., como los manejadores de cache.

- *Seguridad:*

  La seguridad no es simplemente tener un modulo de autenticación en la aplicación en donde se define una ventana de ingreso al sistema, en la que cada usuario escribe su login y contraseña para poder iniciar sesión en la aplicación. La seguridad implica perfilar usuarios para que solo tengan acceso a los módulos o funcionalidades que necesitan, es evitar inyecciones de código a través de los formularios, encriptación de la información a transmitir, entre otros. Por ejemplo, para encriptar la información importante, existen frameworks de transmisión de datos encriptados y para la perfilación la arquitectura puede definir filtros los

cuales son accedidos cada vez que se solicita un recurso para validar o autorizar el uso del recurso al usuario que lo quiere acceder.

- *Mantenibilidad:*

  Con este concepto, se busca que el proceso de mantener y ajustar una aplicación software generada sea una labor fácil y rápida para los desarrolladores, una aplicación ordenada, que siga los estándares definidos por el entorno de desarrollo y que este bien documentada, asegura que esta labor sea menos engorrosa que una aplicación que no aplique lo anteriormente descrito. Por lo anterior, podemos decir que el hecho de la definición de una arquitectura de referencia para las aplicaciones a generar, asegura que los desarrolladores al momento de realizar un ajuste sobre la aplicación, tengan todo el panorama de lo que se verá afectado y de lo que deberán modificar.

- *Interoperabilidad:*

  Esta característica es vital que sea soportada por todas las aplicaciones software que se generan en la actualidad, ya que muchos de los sistemas creados actualmente deben comunicarse con aplicaciones ya existentes, para aprovechar la reutilización de aplicaciones y funcionalidades ya desarrolladas. SOA (service oriented architecture) es un ejemplo de arquitectura que provee interoperabilidad. Para mayor información acerca de SOA, ver el ejemplo del apartado *Arquitectura de referencia* de los conceptos relevantes del marco teórico.

Importancia de la arquitectura de referencia

Debido a las constantes reconstrucciones de servicios y componentes comunes en un software se genera la necesidad de establecer una arquitectura de referencia que permita dar solución a los problemas descritos de una manera confiable y rápida ya que en ésta están predefinidos los componentes para un dominio determinado.
Adicional a esto la arquitectura de referencia colabora con el gobierno de las arquitecturas de aplicaciones que se generan dentro de una organización ya que establece un marco sobre el cual trabajar y provee estructuras reutilizables y reglas, las cuales reducen el tiempo de desarrollo y de realización al igual que aumentan la coherencia en el diseño de las aplicaciones, lo que permite reducir la curva de aprendizaje para los desarrolladores, adicionalmente incrementa la calidad y decrementa el costo del desarrollo.

Problemas comunes que resuelve la arquitectura de referencia

- Problemas recurrentes, es decir puede ser reutilizada en repetidas ocasiones, en numerosos proyectos para dar solución a problemas con características comunes.
- Mantenimiento, debido a la reutilización de componentes y servicios comunes.
- El tiempo al momento de hacer pruebas para dar solución de un problema específico.
- Documentación y descripción de la solución que da la arquitectura al problema específico, como debe utilizarse y en qué momento.
- Ambigüedades, ya que tiene un mando unificado y ampliamente definido.
- Trazabilidad, ya que la arquitectura indica cómo y en qué etapa del proceso se toman las decisiones o transformaciones que se deben modificar.

Ventajas para la organización

- Bajos Costos en la construcción y mantenimiento de software.
- Definición de tiempos en los proyectos más aproximados.
- Construcción de sistemas homogéneos con estándares similares.
- Reducción de riesgos.

Ventajas para los clientes

- Obtención rápida de los beneficios.
- Costos controlados.
- Rendimiento confiable.
- Menor tiempo de implementación.

Ventajas técnicas

- Seguridad, debido a que la arquitectura protege los datos y la infraestructura.
- Orientación, reglas en el diseño e implementación.
- Gobernabilidad: es la capacidad de administración, facilidad en la supervisión y detección de errores.
- Proporciona una base para aplicaciones que funcionan bajo alta disponibilidad.

Ejemplo:

Arquitectura de referencia SOA (Arquitectura Orientada a Servicios)
La arquitectura de referencia SOA contiene en su interior los procesos del negocio y los servicios al igual que su relación entre ellos y con el mundo externo.

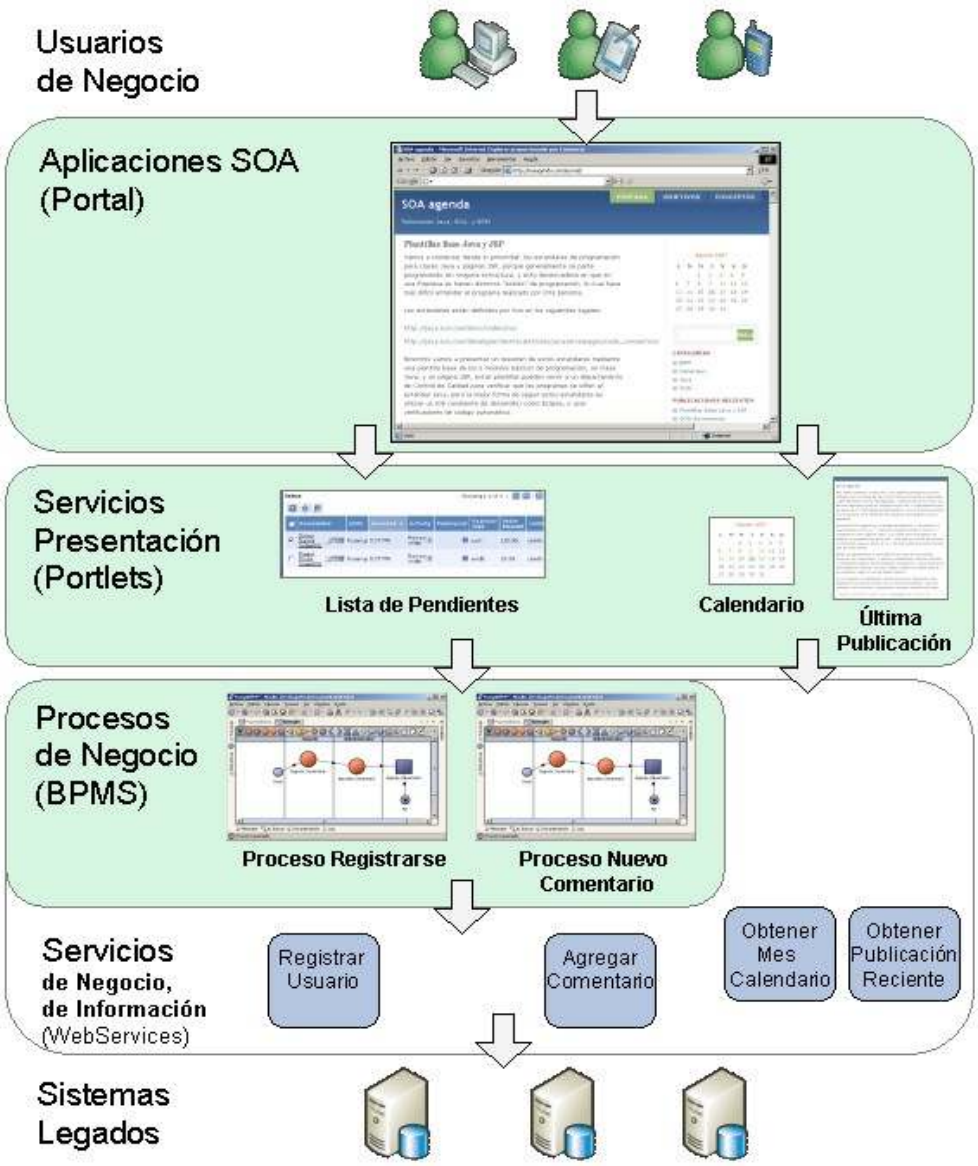


Ilustración 8: Arquitectura de Referencia SOA.

Componentes de la arquitectura SOA

- Usuarios de Negocio: Usuarios de las aplicaciones.
- Aplicación SOA y Portal: Es la aplicación SOA que contiene todos los componentes comunes, son reutilizables y fáciles de de ajustar.
- Servicios de Presentación: Son los componentes reutilizables.
- Procesos de Negocio: Procesos que incorporaran las tareas del participante con los servicios

- Servicios de Negocio: Componentes funcionales del Negocio.
- Sistemas Legados: Son los sistemas que existen en la empresa pero que no están bajo el sistema de orientados a los servicios.

4.2. DIFERENCIAS ENTRE FÁBRICAS DE SOFTWARE

El término “Fábricas de software” puede ser muy ambiguo ya que en la literatura mundial se encuentran dos conceptos que aunque tratan el mismo tema (construcción de software), son totalmente diferentes.

“Fábrica de software”, puede referirse a una empresa u organización que se encarga de la construcción de software, muy comúnmente software genérico, en la que se identifican varios niveles organizacionales, líneas de producción compuestas por una serie de analistas y estrategias de desarrollo soportadas por los modelos de calidad ISO o CMMI. Estas fábricas de software utilizan diferentes metodologías y procesos de desarrollo de software en el que éste se puede dividir en diferentes etapas y fases como el RUP (Proceso unificado de desarrollo de software).

La otra forma de ver una “Fábrica de software”, es como una aplicación funcional (software) la cual dependiendo de una serie de datos de entrada, con unos procesos internos bien definidos y una arquitectura de referencia base, arroja como resultado una aplicación o mínimamente la base de una aplicación. Estas fábricas de software son muy útiles ya que ayudan a tener control en todo momento de las bases y estándares que se quieran implementar tanto en una empresa de desarrollo de software como en un área de negocio de una compañía que si bien no fábrica software como medio de lucro, lo hace para soportar todos sus procesos.
Este concepto de fábrica de software es en el que se enfocará este documento.

4.3. DEFINICION DE FÁBRICA DE SOFTWARE

El concepto de fábricas de software o “Software factory” no es muy conocido en la actualidad, he aquí algunas definiciones:

“Enfoque de desarrollo de aplicaciones en el que confluyen el desarrollo basado en componentes, el desarrollo dirigido por modelos y las líneas de producto software.”
Jack Greenfield y Short en el año 2003.

“Una fábrica de software es una colección estructurada de los activos relacionados con el software. Cuando una fábrica de software esta instalada en un ambiente de desarrollo, ayuda a los arquitectos y a los desarrolladores predecible y eficientemente a crear aplicaciones de alta calidad.”
Microsoft patterns and practices.

“Una línea de producto de software utilizada para producir miembros de una familia de producto mediante la configuración de herramientas extensibles utilizando una plantilla de software basada en un esquema de software.”
Jack Greenfield es un Arquitecto del equipo Enterprise Frameworks and Tools de Microsoft Corporation.

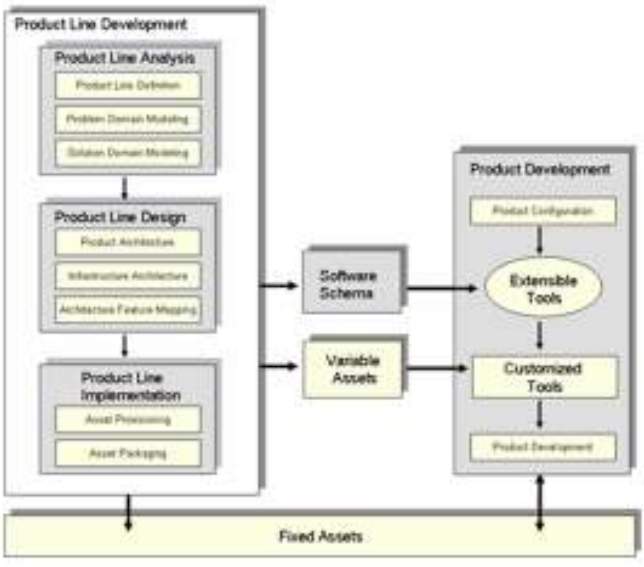


Ilustración 9: Fábrica de Software definida por Jack Greenfield.

Lo que se quiere con el concepto de fábricas de software, es la construcción de una línea de producción (haciendo la analogía con una línea de producción o

ensamblaje bien sea de autos o de electrodomésticos) la cual se encuentra definida dentro de un dominio establecido, que permita agrupar todo un conocimiento para dar una solución adaptable a éste.

Una fábrica de software, es una aplicación software funcional, la cual por medio del uso de modelos de datos, diseños de aplicaciones y en general cualquier tipo de metadata, utilizando una serie de plantillas que definen la estructura o arquitectura que se desea tener en las aplicaciones, genera una aplicación o mínimamente la estructura de esta, facilitando la tarea de los diseñadores, desarrolladores, testers, y todos los implicados en el desarrollo de aplicaciones. Esto también permite que mucha de la complejidad de la etapa de implementación del producto sea oculta por la fábrica de software y al mismo tiempo permita que el producto sea abierto al cambio o a modificaciones específicas que se necesitan desarrollar.

Los componentes fundamentales de las fábricas de software son los patrones, frameworks, arquitectura de referencia, mejores prácticas, modelos que abarcan todas las diferentes etapas por las cuales un software debe pasar para la automatización de los procesos y que hacen posible crear un esquema sobre el cual se podrá realizar el desarrollo y el mantenimiento de productos en particular.

Un ejemplo claro es un rompecabezas, todas las piezas deben estar en su lugar para poder formar la imagen, es lo mismo en la fábrica, todos sus componentes como lo son los patrones, frameworks, plantillas etc., deben funcionar de forma sincronizada de manera que cada una de las partes cumplan su función y que en conjunto se obtengan los resultados esperados.

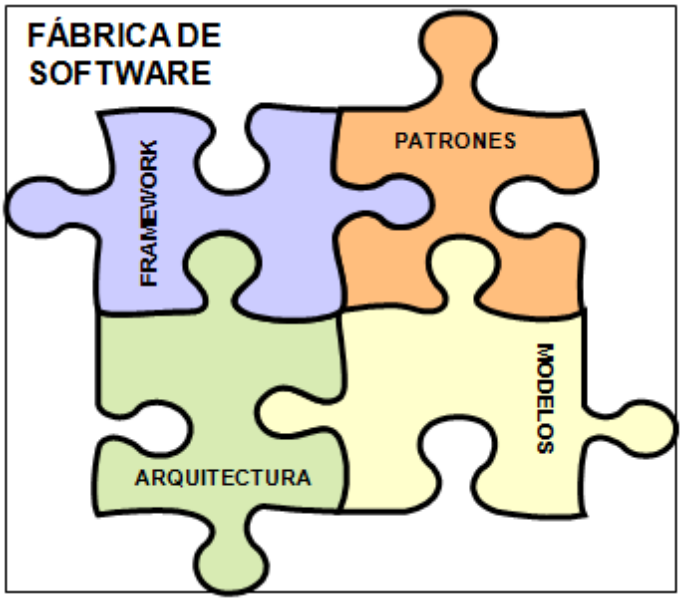


Ilustración 10: Componentes de la Fábrica de Software

Las fábricas de software son también conocidas como “factorías de software”.

Estas fábricas de software buscan la reutilización de código fuente, reducir los tiempos de desarrollo, mantenimiento, certificación y despliegue de las aplicaciones, ya que cuando se ha realizado un proceso de desarrollo al software generado por la fábrica, no es necesario re-hacerlo, y simplemente faltaría desarrollar, mantener, certificar y desplegar las funcionalidades nuevas o especificas que necesita cada aplicación. También se encargan de crear la mayoría del código fuente necesario en una aplicación. Este código puede corresponder no solo a funcionalidades genéricas de las aplicaciones que se generan en una empresa, sino también para la implementación de patrones de diseño, como la conexión a la base de datos, las integraciones con otras aplicaciones, el manejo de logs, cache, entre otros.

Para la construcción de fábricas de software, se deben dar a conocer dos temas que ayudarán a entender con mayor facilidad este concepto.

El primer elemento que se menciona es el de *Esquema de Fábrica de Software*. Para que este concepto quede claro se debe definir en primera instancia que es un esquema. Un esquema según la real academia de la lengua española “… es una representación gráfica o simbólica de cosas materiales o inmateriales. También, es una idea o concepto que alguien tiene de algo y que condiciona su

comportamiento.”. Otra definición para esquema es “… es una forma de analizar, mentalizar y organizar los contenidos de un texto. Se trata de expresar gráficamente y debidamente jerarquizadas las diferentes ideas del contenido para que sea comprensible de un solo vistazo”.

Para enfocar más la definición de esquema a las fábricas de software, se puede decir que es la manera de realizar un análisis detallado de un tema específico, el cual se intenta jerarquizar en niveles y definir una estructura con varios elementos que puedan acoplarse entre si y llevarse a una representación gráfica global del mismo.

Luego de haber definido que era un esquema, ahora se va a definir que es un esquema de fábrica de software. Un esquema de fábrica de software es un documento que permite categorizar y describir aquellos elementos o artefactos que permiten construir y dar mantenimiento de un sistema. Algunos de estos elementos son un documento XML, modelos, archivos de configuración, código fuente, scripts SQL, entre otros. El esquema además muestra una organización y define una relación entre los artefactos permitiendo mantener una consistencia entre ellos.

En el artículo de Microsoft “Fábricas de Software: Ensamblando Aplicaciones con Patrones, Modelos, Marcos y Herramientas”, se define de la siguiente manera “… un esquema de fábrica de software es una plantilla que describe los miembros de una familia de producto de software”.

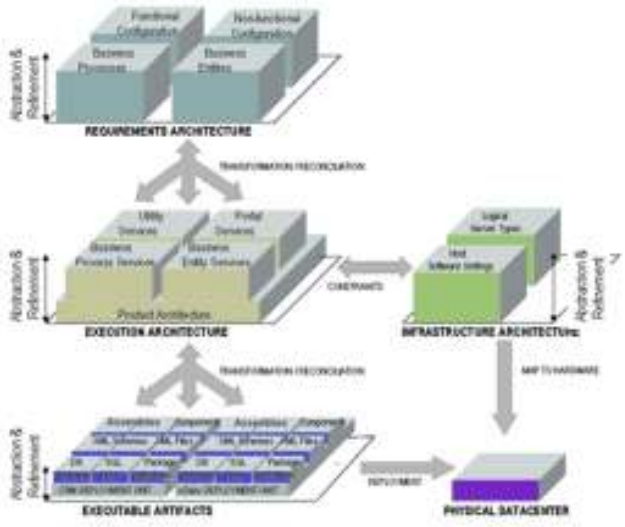


Ilustración 11: Esquema de Fábrica de Software de Jack Greenfield.

La ilustración 11 muestra un esquema de fábrica de software con los artefactos y las relaciones que existen entre ellos.

Un esquema de fábrica de software debe ser flexible al momento de utilizarlo en la construcción de una familia de productos que es para lo cual ha sido creado.
El esquema debe tener asociado partes fijas como variables, donde las partes fijas definen las características comunes que van a poseer cada uno de los productos que van a pertenecer a la misma familia, y por otro lado las partes variables son aquellas propiedades o características que cada producto en particular debe cumplir, en otras palabras, aquellos atributos que va a distinguir a un producto de otro, así pertenezcan a la misma familia.

El segundo elemento a mencionar es el de *Plantilla de Fábrica de Software*. Para dar mayor claridad a este concepto, se va a dar una definición de plantilla. Una plantilla es " … un medio o un instrumento que permite guiar, portar o construir un diseño o esquema predefinido".
Una plantilla agiliza el trabajo de realizar copias similares de un producto, siendo esta un punto de partida o una idea que es aproximada a lo que se requiere hacer.
Para dar una definición dentro del contexto de fábricas de software se podría decir que una plantilla es un diseño predefinido para el desarrollo de un producto software, donde se encuentran incluidos y relacionados aquellos elementos que son relevantes para la creación del mismo.
Una plantilla de fábrica de software es aquella en donde se empaquetan todos los activos necesarios para el desarrollo de un producto como son el esquema de fábrica de software, un conjunto de patrones definidos para la creación del producto, arquitecturas definidas, marcos, herramientas, entre otros.
Además, incluye código fuente y metadatos, los cuales permiten que sean usados por herramientas de desarrollo extensibles como los IDE, permitiendo que el desarrollo de cada uno de los productos de la familia sea un proceso más automatizado y de fácil mantenimiento. Microsoft la llamó plantilla de fábrica de software porque configura las herramientas necesarias para producir un tipo específico de aplicación software, haciendo la semejanza con uno de sus productos como Microsoft Word donde se puede configurar la herramienta para que se pueda crear un documento que sea especifico para ciertas actividades.
Al igual que un esquema de fábrica de software, la plantilla se puede ajustar para desarrollar un producto en especifico en el cual se adicionan, modifican o se eliminan activos que se han definido en esta, logrando así que las especificaciones "únicas" del producto se puedan cumplir.

4.4. ETAPAS DEL PROCESO DE GENERACIÓN EN LAS FÁBRICAS DE SOFTWARE

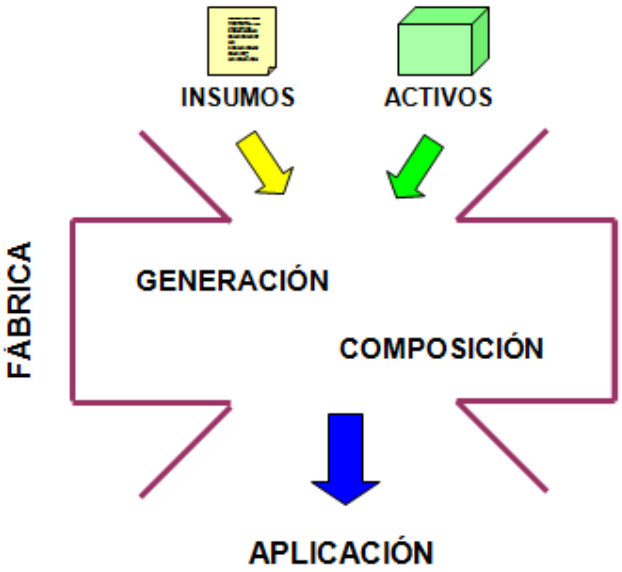


Ilustración 12: Esquema General de la generación de código de una Fábrica de Software.

En el proceso de generación de aplicaciones con la fábrica de software, se identifican dos etapas fundamentales para llevar a cabo este proceso, las cuales se describen a continuación:

4.4.1. Composición

En esta etapa, la fábrica de software define una serie de activos, los cuales componen una base de la aplicación a generar. Esta base en su mayoría cumple con las características no funcionales que debe tener la aplicación como son logger, seguridad, manejo de excepciones, entre otros. Para estos activos, hay que tener en cuenta factores como la compatibilidad de versiones y lenguajes de programación ya que pueden generar problemas e inconsistencias en las aplicaciones construidas.

La línea de producción en esta etapa se comporta de la siguiente manera:

- Como los activos son archivos físicos previamente generados y compilados, la fábrica copiara y pegara estos archivos en la ruta especifica de la aplicación a generar donde estarán las librerías.

- Luego de preparar los activos para que puedan ser utilizados por la aplicación, en la siguiente etapa que es la de generación, la fábrica agrega el código fuente necesario para hacer uso de las funcionalidades que los activos proveen.

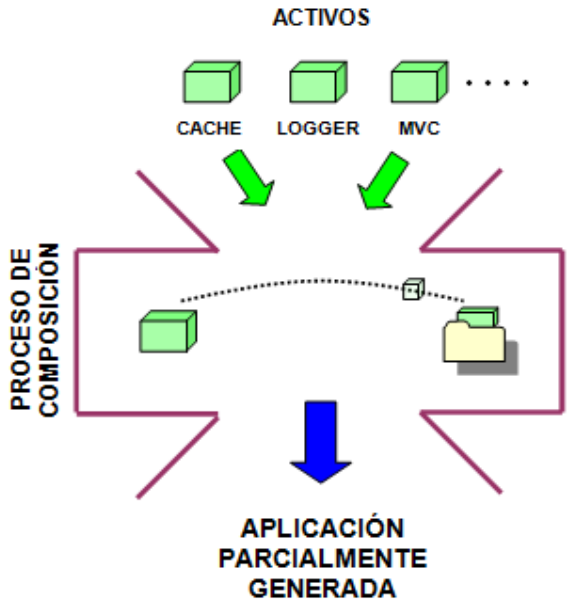


Ilustración 13: Etapa de composición del proceso de generación de una Fábrica de Software.

4.4.2. Generación

Esta etapa es aquella en la cual la fábrica de software crea las aplicaciones generando automáticamente su código fuente, utilizando como base una arquitectura de referencia y los insumos proporcionados por los diseñadores de la aplicación a generar, esto quiere decir que la fábrica tiene autonomía e inteligencia para decidir que archivos y funcionalidades debe crear de acuerdo a lo solicitado por el desarrollador.

Los insumos utilizados en esta etapa son muy importantes ya que definen la estructura de la aplicación a generar e ingresan la información necesaria para que la fábrica funcione.

Los insumos más comunes a utilizar en esta etapa son:

- Metadata: esta metadata son tipos de datos genéricos generalmente XML y XMI, que describen el diseño de una aplicación, generalmente los siguientes artefactos de diseño:
    - Diagrama de clases.
    - Modelo de datos (Diagrama entidad-relación).
    - Descriptores de servicios Web e interfaces de comunicación (WSDL).
- Prototipos (Plantillas HTML).
- Arquitectura de referencia (esquema de la fábrica de software).
- Patrones de software.
- Plantillas de código fuente.

Aquí, la fábrica se basa en las plantillas de código fuente definidas, para generar las clases y vistas de una aplicación que son descritas por la metadata ingresada (Diagrama de clases, modelo de datos, prototipos, etc). Además, si está integrada con una base de datos en particular puede generar todas las tablas necesarias, sino genera los scripts SQL necesarios para crear todo el modelo de datos.

Para dar una idea global de esta etapa, imaginemos una línea de producción en la que una serie de robots están programados para coger cada parte del objeto y ensamblarla en el punto correcto. Dichos robots en este caso son procesadores de texto con unas plantillas de código fuente específicas, y cada parte del objeto es la metadata, entonces el generador de código le dice a los robots que plantilla utilizar y que nombres definir para las variables que encuentre en la plantilla, esto lo hace leyendo la metadata ingresada, analizándola y definiendo que objetos necesita la aplicación para funcionar.

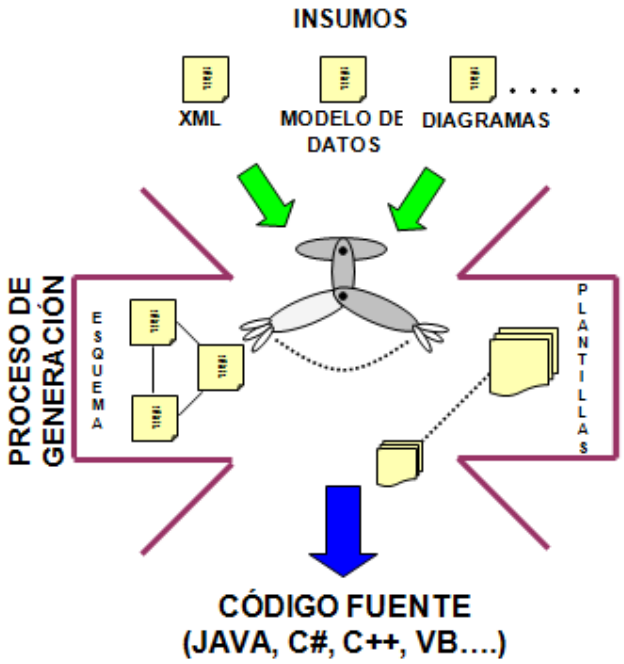


Ilustración 14: Etapa de Generación del proceso de generación de una Fábrica de Software.

4.5. DIFERENCIAS ENTRE MDA Y FÁBRICAS DE SOFTWARE

Como vimos anteriormente, las fábricas de software surgen a través del desarrollo de software dirigido por modelos, pero en este caso también surge otra iniciativa la cual se conoce como MDA.

Para lograr hacer una diferenciación entre estos dos conceptos o metodologías de desarrollo existentes, se debe hacer una pequeña referencia al concepto de MDA.

MDA como las siglas lo muestran es una arquitectura dirigida por modelos (en ingles Model Driven Architecture) la cual está enfocada en el en el uso de las técnicas y métodos que aplica el DSDM (Desarrollo Software Dirigido por Modelos). Es un acercamiento al desarrollo de software de manera eficiente propuesto por el *Object Managment Group* (OMG). MDA provee una serie de guías que ayudan a estructurar especificaciones de un aplicativo software y las expresa con base en modelos.

Luego de haber presentado una pequeña introducción a MDA, ya se pueden mencionar las diferencias que existen en los dos enfoques.
MDA es una solución adecuada cuando se trata de manejar una independencia entre el domino del problema que se está tratando y su implementación en plataformas o tecnologías específicas. Por otro lado el enfoque de fábricas de software es apropiado cuando un grupo de personas resuelven construir sistemas que cumplen con características similares y/o cuando se desea trabajar bajo un dominio determinado.

Una de las posibles desventajas que tiene MDA frente a las fábricas de software es que no proporciona una serie de indicaciones adecuadas de cómo debe ser el diseño de un sistema software, lo que propone son recomendaciones sobre la generación de código para que sea un proceso que se lleve a cabo usando los modelos adecuados, además no se centra en las plataformas donde se van alojar los sistemas sino que los trata de manera independiente. En este caso las fábricas de software proponen como debe ser el desarrollo de software, recomendando varias técnicas ya existentes como el uso de líneas de producto, patrones de diseño, marcos de implementación, arquitecturas, etc.

Una de las mayores ventajas que provee MDA frente a las fábricas de software son los beneficios que otorga en un periodo de tiempo menor, mientras que las fábricas de software al principio sus costos son más elevados debido a la dedicación que se requiere para el desarrollo de un esquema de fábrica de software específico para el dominio en el cual se desea construir, sin embargo los costos al aplicar esta metodología se van reduciendo ya que los productos que se desarrollan son similares, convirtiendo a las fábricas de software en una estrategia de modelado rentable a largo plazo.

Por otro lado una de las ventajas que tiene las fábricas de software sobre MDA es que esta integra muchas áreas de la ingeniería de software que desde hace mucho tiempo han sido investigadas y han progresado continuamente, lo que permite que sean puestas en práctica como por ejemplo los patrones de diseño de software. Esto le ayuda a la fábrica de software a no empezar desde cero, sino que hace una integración y uso de todo el conocimiento que ya ha sido validado por expertos en cada área, obteniendo un software de calidad libre de errores.

Las ventajas mencionadas de un enfoque con respecto al otro, son algunas de las tantas que existen, sin embargo dan claridad de lo que difiere adoptar una de los dos enfoques; además estas diferencias ayudan a tomar una decisión de cuál es la mejor práctica a seguir, dependiendo del problema al que se le quiere dar solución.

4.6. EJEMPLOS DE FÁBRICAS DE SOFTWARE

En la actualidad existen herramientas que ayudan a desarrollar las aplicaciones software a partir de una fábrica de software. Estas aplicaciones la mayoría son de la empresa Microsoft como son Smart Client Software Factory, Web Cliente Software Factory, Mobile Client Software Factory y Web Service Client Software Factory. Estas fábricas de software trabajan sobre el ambiente de desarrollo Visual Studio .Net otorgando mayor funcionalidad en la creación de productos software.

- **Smart Client Software Factory:** es una herramienta que provee un conjunto integrado de guías para asistir o ayudar a los arquitectos y desarrolladores a crear una combinación de aplicaciones inteligentes. Las características que poseen los productos generados por esta fábrica son:
  - Posee una interfaz de usuario que toma las ventajas del desarrollo de aplicaciones de escritorio sobre Microsoft Windows
  - Conecta múltiples sistemas back-end para el intercambio de datos.
  - Muestra información desde diversas fuentes a través de una interface de usuario integrada, así los datos se pueden observar como si se estuviera viendo en el sistema fuente.
  - Toma la ventaja de los recursos de almacenamiento y procesamiento local para habilitar operaciones durante un periodo de no conexión o de conexión intermitente a la red.
  - Posee fácil instalación y configuración.

- **Web Client Software Factory:** es una herramienta que provee un conjunto de guías para arquitectos y desarrolladores que construyen aplicaciones empresariales basadas en Web. La fábrica incluye ejemplos, código reutilizable y un paquete orientado, el cual automatiza las principales tareas de desarrollo dentro de Visual Studio.

  La ventaja de usar esta herramienta es que los desarrolladores pueden crear una combinación de aplicaciones web compuestas por desarrollos y módulos de despliegue separados.

- **Web Service Software Factory:** es una colección de recursos integrada, diseñada para ayudar rápida y consistentemente la construcción de servicios web que se adhieren a los patrones de diseño y arquitectura conocidos.

- **Mobile Client Software Factory:** es una herramienta que provee una guía integrada para ayudar a los arquitectos y desarrolladores a crear una línea de negocios basada en aplicaciones de Windows Mobile que interactúan con sistemas back-end sobre redes tales con WiFi y GPRS.

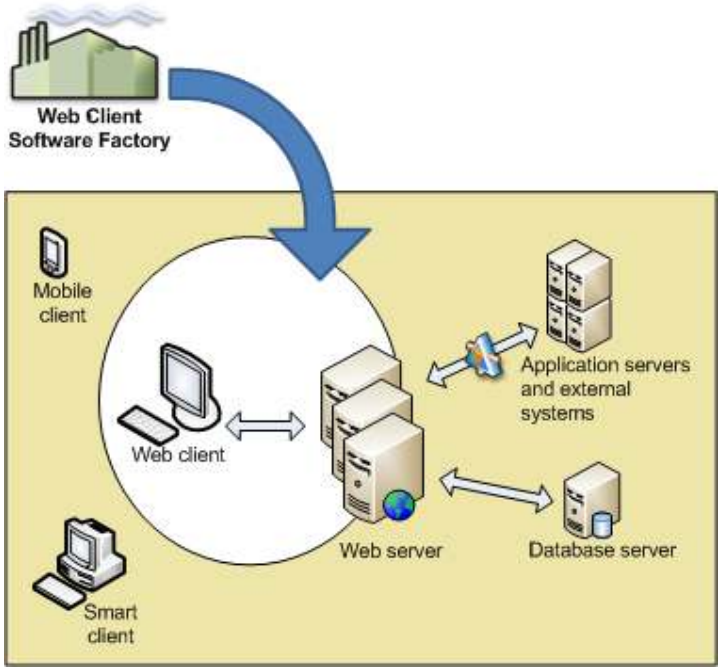


Ilustración 15: Esquema de trabajo de Web Client Software Factory de Microsoft.

5. GUIA METODOLÓGICA

Para comenzar con la creación de una fábrica de software, se debe tener claridad en los conceptos que han sido definidos y explicados en este documento, para que así no haya inconvenientes a la hora de concebir la fábrica de software.

Ésta guía metodológica está enfocada en el desarrollo de software, por lo cual se deben tener presente otros aspectos transversales al proceso de desarrollo que no se mencionarán en la guía.

A continuación se presentarán un resumen de los pasos propuestos a seguir para la construcción de la fábrica de software y posteriormente de ampliará la definición de cada uno.

PASO	BREVE DESCRIPCIÓN
Modelado del Negocio	Recopila la información de todo el modelo de negocio haciendo que sea identificable las posibles necesidades que se puede suplir con aplicaciones software generadas por la fábrica.
Dominio del Problema	Analiza el modelo del negocio identificando y especificando cuales son los problemas que se presentan y se quieren solucionar con la creación de la fábrica de software.
Dominio de la Solución	Analiza los problemas identificados y define las posibles soluciones que se pueden emplear.
Diseño de la solución	Se seleccionan las soluciones óptimas para cada uno de los problemas identificados haciendo que sean posible implementarlas genéricamente por la fábrica de software, mediante la definición de la arquitectura de referencia, los aspectos no funcionales y el esquema de la fábrica de software.
Seleccionar un generador de código	Se estudia y se selecciona en el mercado el generador de código que mejor se acople a las necesidades anteriormente identificadas

Creación de las plantillas de la fábrica	De acuerdo al generador de código seleccionado se genera las plantillas que describen la arquitectura de las aplicaciones que va a generar la fábrica.
Implementación de la línea de producción	Se construye el generador de aplicaciones, el cual maneja la lógica de creación de código fuente basándose en las plantillas y el generador de código seleccionado.
Pruebas de la fábrica de software	Como la fábrica es una aplicación software debe pasar por una etapa de pruebas que certifique su correcto funcionamiento.
Mantenimiento de la fábrica de software	La fábrica como todo software, debe permanecer en continuo crecimiento haciéndola cada vez más madura y confiable.

Tabla 2: Resumen de los pasos propuestos en la guía.

ESPECIFICACIÓN DE LOS PASOS PROPUESTOS PARA LA CONSTRUCCIÓN DE UNA FÁBRICA DE SOFTWARE

PASO 1: Modelado del Negocio

Para hacer un desarrollo adecuado de una fábrica de software, se debe tener presente realizar un modelo de negocio bien elaborado, el cual permite conocer con detalle el problema al que se le quiere dar solución. Para esto se debe hacer un estudio profundo de la estructura y funcionamiento de la organización, respondiendo a unas preguntas claves que ayudan a identificar y conocer las posibles necesidades que esta tiene. Algunas de estas preguntas son:

- ¿Cuál es el mercado en el que se mueve la organización?
- ¿Cómo define y diferencia los productos que esta ofrece?
- ¿Cómo crea utilidad para sus clientes?
- ¿Cómo consigue y mantiene a sus clientes?
- ¿Cuáles son las estrategias publicitarias y de distribución que la organización usa para dar a conocer sus productos?
- ¿Cómo están definidos los procesos en la organización?
- ¿Cómo realiza la gestión de los recursos que están a su disposición?

- ¿Cómo consigue su beneficio con respecto al mercado en el cual se desenvuelve?

Luego se debe analizar cada una de las respuestas que se obtuvieron a partir de las preguntas elaboradas previamente, con el fin de poder tener una perspectiva general y específica del negocio. Para ello se debe estructurar cada uno de los componentes obtenidos (clientes, procesos, entre otros) y plasmarlos en un diagrama que tenga un orden lógico, para así poder entender más fácil cuales son las relaciones que existen entre ellos y abstraer cuales pueden ser las posibles necesidades que se quieren resolver con la fábrica de software.

Además se debe realizar un glosario con las palabras clave del negocio, permitiendo que se hable el mismo lenguaje, que se pueda entender a que se dedica y que hace la organización, y a la hora de construir la fábrica de software no se pierda el enfoque de lo que se quiere solucionar.

Para facilitar el reconocimiento en el modelado de la empresa se recomienda realizar un diagrama de procesos que describa cada una de las áreas y los procesos que comprende la organización. Este diagrama apoya la construcción de la fábrica de software, ya que de él se puede obtener de manera general los tipos de aplicaciones a generar por la fábrica, como por ejemplo, aplicaciones financieras, de apoyo a la logística, de puntos de servicio, entre otras.

PASO 2: Dominio del Problema

Después de realizar una definición sobre el modelo del negocio de la empresa, se debe realizar un estudio del mismo para identificar posibles problemas que pueden ser solucionados por la fábrica de software al igual que se debe evaluar el dominio del problema en el cual éste se desarrolla.

Para identificar cual es el problema que se está presentando es necesario examinar a fondo todo el modelo del negocio, para así en el momento en el cual surja una dificultad saber que se está creando una necesidad que debe ser suplida, es decir se presenta diferencia entre lo que se está observando y el ideal de esa realidad.

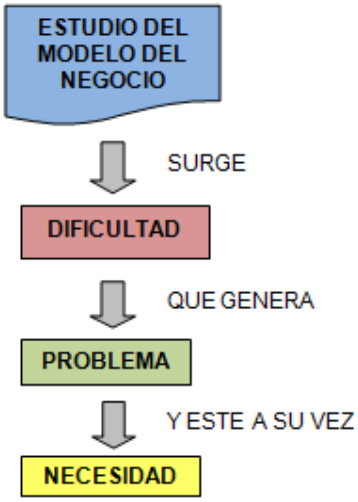


Ilustración 16: Análisis del dominio del problema

En la literatura mundial se pueden encontrar varias preguntas que ayudan a identificar más fácilmente cual es el problema que se presenta:

- Quién?:
  Quien dice que esto es un problema?
  A quien afecta?
  Quien causó o está causando el problema?

- Qué?
  Qué sucedió o sucederá?
  Cuáles son las consecuencias para otras personas?
  Qué es lo que no está funcionando según se desea?

- Cuándo?
  Cuando Sucedió?
  Cuando ocurrió por primera vez?

- Dónde?
  Dónde está ocurriendo el problema?
  Dónde tuvo o tendrá efecto?

- Por qué?

Por qué constituye esto un problema?
Por qué ocurrió?
Por qué se requiere una respuesta ahora mismo?

- Cómo?
  Cómo debiera estar funcionando el proceso?
  Cómo están enfrentando este problema?
  Cómo sabe que es un problema?

Después de tener identificado el problema en cuestión debemos estudiar el dominio en el cual éste se desarrolla para definir un límite o un alcance en el cual se debe desarrollar dichos problemas, para esto se debe identificar los objetos que hacen parte del problema, es decir conceptos que tienen propiedades, atributos y/o relaciones con otros objetos del dominio, al igual que se deben definir las facetas de éste que ayudan a representar las relaciones espaciales, en otras palabras, el orden, la jerarquía que deben llevar los objetos dentro del dominio.

Este paso permite identificar, luego del análisis realizado a los problemas observados en el modelo, las funcionalidades críticas y necesarias que debe tener toda aplicación generada por la fábrica en el dominio analizado, por ejemplo, las auditorías a las aplicaciones, seguridad en las transacciones, control de cambios en la información, forma de comunicación de las aplicaciones, almacenamiento de registro de errores, persistencia de la información en memoria, etc., haciendo que estas funcionalidades se conviertan en requisitos que debe resolver la fábrica de forma genérica para todas las aplicaciones.

PASO 3: Dominio de la Solución

Teniendo en cuenta el dominio del problema a resolver, se debe hacer un análisis profundo de éste para definir una solución que supla las necesidades que fueron generadas, al igual que mejore la calidad en el resultado deseado.

Luego de tener identificados los problemas se deben definir cuáles de estos va a solucionar la fábrica de software identificando claramente si van a ser solucionados genéricamente o deben ser implementados de acuerdo a las diferentes necesidades del negocio. Además de esto, se debe dejar en claro que problemas no va a abordar la

fábrica para así buscar soluciones externas o a la medida que supla las necesidades que cada aplicación a generar pueda tener.

Existen varias maneras para realizar un análisis de la solución de problemas, aquí se proponen algunas que le pueden ayudar en este proceso:

Metodologías de Apoyo

- *Lluvia de Ideas:*
  Son sesiones en las cuales se reúnen un grupo de personas a quienes se les da la oportunidad de opinar o sugerir sobre un determinado asunto que se trabaja.

- *Diagrama de Paretto:*
  Gráfico que está compuesto por barras verticales ordenadas por importancia. Éstas barras representan datos específicos correspondiente al problema, que ayuda a centrar la atención y esfuerzo en los problemas realmente importantes.

Teniendo en cuenta las decisiones tomadas en este dominio de acuerdo al problema que se quiere solucionar y a los servicios que ofrece el negocio, se debe definir cuáles son los tipos de productos que la fábrica de software va a generar como aplicaciones web, móviles, de escritorio y/o Web Services.

PASO 4: Diseño de la solución

Luego de conocer a profundidad el negocio entendiendo claramente sus procesos internos, sus fortalezas y debilidades, e identificar puntualmente sus necesidades para la construcción de software, se debe abarcar el diseño de la solución, basándose en el paso anterior, el cual entrega como insumo los requerimientos que son la base para la implementación de cualquier aplicación software. Se debe tener claro que, como las fábricas de software son aplicaciones software, su construcción atraviesa por todas las etapas de construcción de software tradicional, por lo que para su creación se debe haber definido unos requerimientos, diseñar la aplicación, implementarla, probarla y mantenerla. Explicar todas las anteriores actividades no es el objetivo de esta guía, ya que hacen parte de la construcción de software tradicional, pero aparte de estas, hay

algunos pasos que son básicos e inamovibles para la construcción de fábricas de software ya que son la base del funcionamiento de estas.

PASO 4.1: Definir la arquitectura de referencia

Definir la arquitectura de referencia es lo mas importante en el momento de la construcción de una fábrica de software, ya que define la estructura de todas las aplicaciones que se generaran con la fábrica, y más aun, define como deben ser implementadas todas las aplicaciones que se diseñen en el entorno en el que se encuentra la fábrica, ya sea una empresa desarrolladora de software o una compañía que produce software para mantener sus procesos de negocio. Todos los analistas, desarrolladores y demás participantes en el proceso de construcción de software tienen que conocer muy claramente la arquitectura, ser capaces de pensar en soluciones de software adaptadas a dicha arquitectura, conocer sus ventajas y debilidades, y las posibilidades de diseño e implementación que esta les proporciona.

Se debe prestar gran atención a la definición de la arquitectura de referencia ya que esta puede modificar el comportamiento de la fábrica de software, por ejemplo cuando generamos una aplicación con la fábrica de software, hay una etapa del proceso de desarrollo que no podemos dejar a un lado y es el mantenimiento, por lo que para esto la fábrica debe regenerar la aplicación, lo que implica recrear todos los archivos y posiblemente una pérdida de la funcionalidad especifica desarrollada. Entonces, aunque evitar este problema no es tarea de la arquitectura de referencia, esta si puede proveer ayuda para esto, mediante la implementación de alguna estructura que facilite la ubicación de código fuente especifico, por ejemplo en los objetos del negocio.

Para una correcta definición de la arquitectura de referencia se deben tener en cuenta los siguientes conceptos:

- *Alta cohesión*
- *Bajo Acoplamiento*
- *Mejores Prácticas de desarrollo*
- *Patrones y Frameworks*

Al momento de definir la arquitectura de referencia, se debe tener presente toda la lógica no funcional que se necesita que ésta soporte, ya que dependiendo de esta lógica la arquitectura debe ser definida de una manera u

otra. A continuación veremos algunas características de lógica no funcional importantes para tener en cuenta:

- *Portabilidad.*
- *Escalabilidad.*
- *Seguridad.*
- *Mantenibilidad.*
- *Interoperabilidad.*

Con lo anteriormente mencionado, se puede observar que la tarea de definir una arquitectura de referencia no es fácil, se debe tener un amplio conocimiento no solo del modelo del negocio para el cual aplicará la fábrica, sino también un conocimiento académico elevado, en el cual se destaca conocer muchos patrones de diseño de aplicaciones, las diferentes ofertas que tiene el mercado en frameworks y sobretodo la capacidad para plasmar e ingeniar las necesidades que plantea el dominio de un problema en tecnología. Las siguientes son una serie de preguntas que son importantes de cuestionar cuando se está definiendo una arquitectura de referencia:

- ¿Nos basaremos de alguna arquitectura existente? ¿Cuál?

- ¿Cuáles son las características no funcionales más importantes que la arquitectura debe proveer?

- ¿Cómo se debe soportar las funcionalidades específicas?
    - Transmitir y traducir errores.
    - Persistir información en memoria.
    - Almacenar logs.

- ¿Qué sucedería si?

- ¿Cómo hacemos si?

Paso 4.2 Definir los aspectos no funcionales de la fábrica de software

Como se ha dicho anteriormente, una fábrica de software es una aplicación software, por lo que se debe definir todos sus requerimientos no funcionales.

Para esto, hay que tener en cuenta la arquitectura de referencia ya que esta puede dictar la solución a algunos de estos requerimientos.

- Lenguaje de programación: como la fábrica de software no necesariamente debe ser portable porque va a ser utilizada en un ambiente especifico, se debe elegir un lenguaje de programación que optimice los recursos de dicho ambiente, por lo general lo más importante es el rendimiento, ya que una de las funciones básicas de una fábrica de software es reducir los tiempos de desarrollo.

- Interfaz para la conexión a la base de datos seleccionada: la fábrica de software no necesita una base de datos para su funcionamiento, pero el producto que va a generar sí la necesita. La fábrica debe conectarse a dicha base de datos para crear el esquema de base de datos recibido a través del diagrama entidad-relación (si es el caso).

Paso 4.3 Definir el esquema de la fábrica de software

Como se vió en el apartado *definición* de la fábrica de software, el esquema define los artefactos que permiten construir un sistema. Para esto, es muy importante definir dentro del proceso de desarrollo cuales son los artefactos de diseño necesarios a implementar, y cuáles de estos se convertirán en los insumos para la generación de la aplicación a través de las plantillas.
El éxito o fracaso de la implantación de una fábrica de software, depende en gran medida de este paso, ya que una fábrica es útil en la medida en la que agiliza y mejora el proceso de desarrollo, y esto se logra gracias a los metadatos que recibirá la fábrica.
El diagrama de clases es el primero y posiblemente por muchas personas único artefacto que se pensaría como insumo para la fábrica, ya que provee los nombres de las clases, sus atributos y funcionalidades, además de la relación entre ellas. Pero hay otros diagramas que con un análisis detallado y una buena definición e implementación en la arquitectura y las plantillas, pueden ayudar a mejorar aún más las aplicaciones generadas por la fábrica como por ejemplo el modelo entidad-relación y el diagrama de secuencia.
Con el modelo entidad-relación, se pueden generar los objetos persistentes, la implementación de los DAO (si es el patrón a implementar), la creación de los

scripts de creación de la base de datos (SQL) y las relaciones que hay entre los diferentes objetos persistentes. Por su parte, el diagrama de secuencia define las dependencias y los llamados a otros métodos creados en otras clases, cuales son los atributos que reciben y qué tipo de atributo retorna.
Los prototipos son otro tipo de diagramas que se pueden utilizar, ya que si bien el diseño visual de las aplicaciones varían, hay formularios o vistas que pueden generalizarse como por ejemplo la administración de tablas maestras de una aplicación, ya que por medio de un prototipo genérico y del modelo de datos, se puede crear una vista la cual permita el CRUD (siglas en ingles de crear, leer, actualiza y borrar) sobre los atributos de una determinada tabla u objeto de negocio.

Luego de definir los diagramas a diseñar, se debe definir y diseñar el código fuente que generara la fábrica por medio de las plantillas. Para esto, hay que cruzar la arquitectura de referencia, los diferentes diagramas a crear y la lógica funcional que debe tener los métodos o funcionalidades a crear automáticamente. Por ejemplo, si la arquitectura de referencia define que para cada objeto persistente se debe tener un DAO para la implementación de las funcionalidades sobre la base de datos y un objeto de negocio con el mismo nombre de la tabla para la interpretación de la información, entonces se debe cruzar esta arquitectura con los diagramas a diseñar para obtener los nombres de las tablas y de ahí sacar el nombre del DAO y del objeto de negocio, para identificar los atributos que tendrá el objeto de negocio y los atributos que interactuarán en el DAO. Además se debe cruzar también esta información con la funcionalidad que se desea proveer para crear en el DAO los métodos de selección, inserción, actualización y borrado sobre dichos atributos y objetos. También, con esto se identifica que se deben tener dos plantillas, una para el DAO y otra para el objeto de negocio que se va a generar.

PASO 5: Seleccionar un generador de código

Luego de haber realizado el diseño de la solución (con su arquitectura y esquema), se debe buscar cual puede ser el generador de código indicado que va a usar la fábrica al momento de generar un producto.
Para ello se debe estudiar varios de los generadores de código existentes en el mercado. Este estudio comprende la facilidad de uso del generador, la complejidad que maneja para su uso, las características o ventajas que provee para ayudar a elaborar un producto software, el tipo de licencia que tiene para su uso (esto depende de la capacidad y las políticas de la organización para el uso de software libre o comercial), y por último la facilidad de adaptarlo a lo que se ha establecido como la solución del problema.
Algunos de los generadores de código que existen en el mercado se han mencionado en el documento (*ver Apéndice B*), lo cual puede ayudar a tomarlos como referencia y decidir cuál es el más apropiado para lo que se está haciendo.

PASO 6: Creación de las plantillas de la fábrica

Después de tener definido el esquema de la fábrica de software es necesario crear las plantillas propias del negocio que van de la mano del generador de código que fue escogido en el paso anterior.
Para la creación de las plantillas se utiliza como base la arquitectura de referencia que fue diseñada previamente ya que esta utiliza los patrones, las guías que son parte fundamental de la arquitectura, para luego, después del proceso de conversión al formato deseado (xmi, xml, etc.) genere una plantilla que el generador de código sea capaz de interpretar y que plasme en la aplicación a desarrollar todas las definiciones que fueron realizadas en la arquitectura de referencia.

Es importante aclarar que estas plantillas varían según la arquitectura de referencia puesto que la arquitectura como tal es la base fundamental de éstas y si no se tiene bien definida la arquitectura, por ende, las plantillas no cumplirán con las características deseadas.

Se deben proveer plantillas que especifiquen las tecnologías sobre las cuales van a ser desarrollados los productos generados por la fábrica como:

- Bases de datos: Es importante definir sobre que motor de base de datos trabajará la aplicación generada por la fábrica, pues al momento de generar automáticamente el código la fábrica debe crear las conexiones con la base de datos.

- Lenguajes de programación: se deben tener diferentes plantillas que definan el lenguaje en el cual va a ser generada la aplicación.

- Interacción con aplicaciones: hay que identificar claramente cuáles son los tipos de interacciones que va a tener la aplicación ya que se deben tener plantillas específicas para cada tipo de interacción, por ejemplo, comunicación por medio de WSDL.

PASO 7: Implementación de la línea de producción

Para la creación de los productos que se quieren obtener a partir de la fábrica de software, se debe definir una línea de producción la cual sea adecuada a lo que se ha definido en los pasos anteriores (para poder entender claramente los siguientes puntos ver punto 4.4 con sus respectivas Ilustraciones).

En este punto es donde se integran cada uno de los activos que hayan sido definidos para la construcción de la fábrica como el(los) framework(s), los patrones a usar, cada una de las capas que hayan sido definidas en la arquitectura de referencia y la relación entre estas, el esquema que se ha definido para esta, el generador de código a usar, la tecnología de desarrollo de software que ha sido seleccionada para su implementación y la definición de datos que se hayan establecido para las plantillas de la fábrica.

Al momento de realizar la implementación en la tecnología seleccionada (lenguaje de programación) y para la herramienta necesaria (IDE si es del caso), es recomendable que si se tienen componentes desarrollados anteriormente y sean posibles de acoplar con lo que se ha definido, se usen para la implementación, permitiendo que el tiempo de desarrollo sea menor.

La línea de producción de la fábrica de software se encarga de hacer el ensamblado de los activos que se han seleccionado para así crear la base del producto software. Se debe comenzar a desarrollar la línea de producción con la tecnología de desarrollo seleccionada, empezando a realizar componentes genéricos y flexibles que sean

fáciles de mantener, a partir del (de los) framework(s) seleccionado(s), permitiendo la reutilización de los mismos en otros proyectos, o haciendo que el desarrollo de la línea sea más fácil.
Luego se hace la traducción a la tecnología seleccionada de la arquitectura de referencia y del esquema que han sido definidos en la etapa de diseño, y acoplarlo a los componentes genéricos que se han desarrollado. En esta fase se debe tener en cuenta el generador de código, estableciendo cuales son los objetos y la estructura de la base que debe generar este para un producto que vaya a hacer ensamblado por la línea de producción.

Es importante aclarar que en el momento de la implementación de la línea de producción, hay que definir cuales son los puntos de extensión sobre los cuales la fábrica identifica que porciones de código son las escritas por los desarrolladores para la creación del producto, respetando esta parte del código para la regeneración del producto.

PASO 8: Pruebas de la fábrica de software

Para llevar a cabo las pruebas sobre la fábrica de software es necesario tener en cuenta lo siguiente:

- El resultado que está arrojando la fábrica de software es el deseado y da solución al problema que fue identificado en un principio.
- La aplicación generada por la fábrica de software sea cargada por el IDE seleccionado y que pueda compilarse sin problemas.
- En el momento de regenerar el código no se pierda la información que ya ha sido integrada en él.

Estas pruebas exigen menos tiempo puesto que muchos de los componentes utilizados en la fábrica ya han sido probados anteriormente en otras aplicaciones y los posibles errores que puedan presentarse son fácilmente identificables si se tuvo una buena definición de la arquitectura de referencia.
Las pruebas más comunes que se realizan sobre las fábricas de software y sus aplicaciones generadas son:

- Pruebas Unitarias: se basan en probar el correcto funcionamiento de cada módulo de la aplicación, esto con el fin de asegurar que todos los módulos que fueron implementados funcionan correctamente por separado.

- Pruebas de Integración: estas pruebas consisten en probar que todos los módulos de la aplicación en conjunto funcionan correctamente.

- Pruebas Funcionales: Esas pruebas se basan en los modelos de pruebas previamente diseñados que buscan evaluar cada una de las funciones de la aplicación.

- Pruebas de Validación: con esta prueba se pretende validar que el software producido cumple con las especificaciones iniciales hechas por los clientes.

- Pruebas de Stress: En esta prueba se pretende asegurar que el sistema funciona como se espera bajo grandes volúmenes de transacciones, usuarios, entre otros.

PASO 9: Mantenimiento de la fábrica de software

Con la fábrica ya desarrollada, probada y en ejecución se deben identificar cambios que han surgido desde el momento en que esta fue concebida con el fin de actualizarla y/o mejorarla.

Como se menciono en el paso 7, la idea de la fábrica de software es que desde el momento de su concepción sea fácil de mantener, para lo cual se crea un espacio que permite escribir código que puede ser reutilizado. En este espacio se debe escribir el código fuente que la línea de producción va a usar, bien sea para optimizar el producto que ya se ha construido o usarlo en los nuevos productos que van a ser desarrollados, ó para actualizar el esquema en el cual la fábrica fue concebida, ayudando a que los cambios necesarios para hacer, no afecten el funcionamiento de la misma y se puedan hacer en un solo lugar.

También permite corregir errores que fueron detectados desde el momento en el cual se ha construido la arquitectura o en la de fase pruebas, haciendo que el proceso de corrección de errores se haga en un tiempo menor.

Hay que tener en cuenta que si se realizan cambios en la arquitectura de referencia, las aplicaciones que han sido generadas por ésta ya no se pueden regenerar, ya que puede causar inconsistencias y pérdida de información.

EJEMPLO PRÁCTICO, APLICANDO LOS PASOS PROPUESTOS EN LA GUÍA

Después de haber realizado la propuesta de los pasos que se deben llevar a cabo para la construcción de las fábricas de software, se pondrán en práctica dichos pasos por medio de un ejemplo simple.
Como se menciono en el alcance de este documento, el ejemplo no tendrá ninguna implementación funcional que se ponga en práctica, por tal motivo, los pasos 7, 8 y 9 no aplican para este ejemplo.

INTRODUCCIÓN DEL EJEMPLO

ARCAN S.A es una empresa de software que se especializa en el desarrollo de aplicaciones a la medida para automatizar los procesos de almacenes de cadena a nivel nacional que se dedican a la venta de ropa masculina y femenina para todas las edades.
Debido a la gran demanda que se ha generado en el mercado para automatizar los procesos, ésta empresa ha decidido construir una fábrica de software, la cual ayudará agilizar su proceso de desarrollo, dándoles un mayor índice de competitividad y efectividad para poder suplir las necesidades de sus clientes en un tiempo más corto.

PASO1: Modelado del Negocio

El mercado objetivo en el cual se mueve las cadenas de almacenes en estudio es la población de diferentes edades organizadas en grupos como bebes, niños, jóvenes, adultos y tercera edad de ambos sexos que desean comprar prendas de vestir.
Buscando satisfacer las necesidades de todos los grupos presentan artículos de diferentes marcas que varían en sus precios. Como valor agregado para sus clientes estos almacenes incentivan la compra por medio de cupones de descuento, rifas, promociones y diferentes actividades dadas a conocer a través de los medios de comunicación como cuñas radiales, comerciales televisivos, correo electrónico y revistas mensuales que circulan por todas las ciudades.

Para poder ofrecer un excelente servicio no solo a sus clientes sino a nivel interno, estas empresas cuentan con las siguientes áreas básicas que permiten el crecimiento continuo de la organización:

- Área Administrativa y Financiera: Encargada de los procesos de facturación, contabilidad y nómina.
- Área de Mercadeo: Esta se encarga de la publicidad para atraer más clientes y la motivación a éstos para mantenerlos.
- Área de compras y ventas: Se encarga del manejo de la adquisición y venta de los productos y del manejo del inventario.

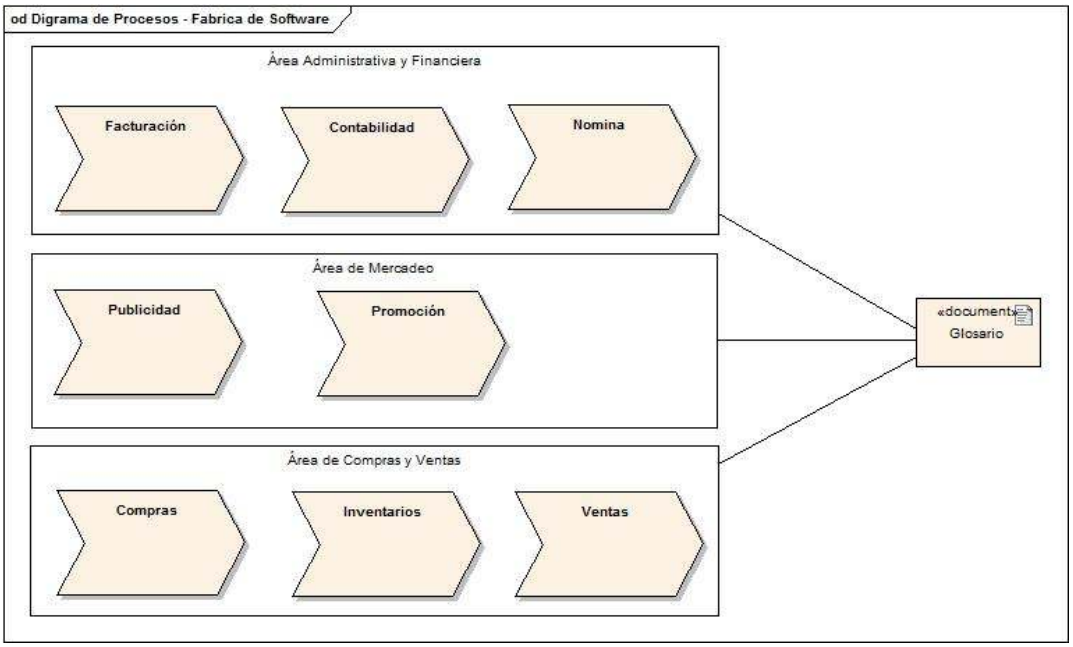


Ilustración 17: Ejemplo del diagrama de Procesos

Glosario de Términos del Negocio:

- Unisex: Ropa que puede ser utilizada por ambos sexos.
- Tallaje: este término se utiliza para referenciar las diferentes tallas que pueden presentar en una prenda de vestir.

PASO 2: Dominio del Problema

Los problemas que han identificados después de realizar un estudio sobre el modelo del negocio y las necesidades expresadas por los clientes, se encuentran las siguientes dificultades que puede solucionar la fábrica de forma genérica:

- Conectividad hacia las diferentes bases de datos que tienen sus aplicativos.
- Necesidad de conectarse a diferentes aplicativos para compartir información.
- Extracción de información relevante por medio de reportes que ayudan a la toma de decisiones.
- Como son almacenes de cadena, se ha observado la necesidad de tener una aplicación centralizada a la cual accedan varios usuarios simultáneos para el manejo de todas las operaciones del almacén.
- Se necesita enviar correos electrónicos a los diferentes clientes de los almacenes para mantenerlos actualizados sobre los productos que se ofrecen.
- Se requiere almacenar registros de las diferentes transacciones que se hagan en las diferentes operaciones del almacén.

PASO 3: Dominio de la Solución

Las aplicaciones a ser generadas por la fábrica, deberán soportar las siguientes funcionalidades que se identificaron en el paso anterior:

- Deben soportar la conexión a diferentes bases de datos, haciéndola flexible a cambios futuros o conexiones a bases de datos de otras aplicaciones.
  La elección de la base de datos dependerá de la necesidad y/o de la capacidad económica que tenga el almacén. Esto no es problema para la fábrica ya que lo que ella genera es la conexión a la base de datos dependiendo de la elección, además, si se quiere que esta genere los scripts de creación, se crean en el lenguaje estándar SQL que es independiente del motor de base de datos seleccionado.

- Deben proveer la comunicación con otras aplicaciones a través de Web Services ya que es la tecnología mas usada en el momento.

.

- Contaran con un generador de reportes en formatos de hojas de cálculo y documentos en PDF.

- Como deben ser aplicaciones centralizadas y deben soportar múltiples conexiones al tiempo, serán generadas bajo plataforma web.

- Deben tener un generador de correos masivos que permitan el envío de publicidad.

- Como se necesita auditar todas las transacciones, la fábrica generará los logs y su administrador para cada aplicación que genere.

- El lenguaje de programación para las aplicaciones generadas por la fábrica de software será Java, porque su licencia es gratuita y es un lenguaje portable para que sea adaptado a cualquier software que se tenga entre los diferentes almacenes.

PASO 4: Diseño de la solución

La arquitectura de referencia en la cual se va a basar la fábrica de software para la generación de productos software para los almacenes es la siguiente:

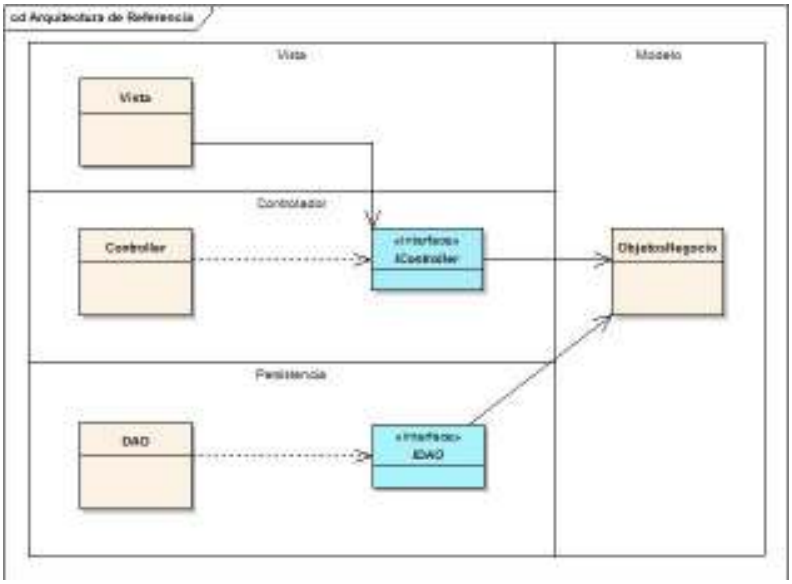


Ilustración 18: Ejemplo de arquitectura de referencia.

En la grafica anterior se puede observar las capas definidas para la arquitectura de este caso:

- La persistencia, maneja todo lo relacionado con la base de datos, es decir la conexión, los métodos tanto básicos como avanzados sobre la base de datos.

- El modelo, tiene los objetos de negocio que son necesarios en la aplicación.

- En el controlador encontramos los delegadores de servicios que controlan el tráfico de información entre el modelo y las vistas.

- La vista, son los formularios o interfaces graficas con las cuales interactúan los usuarios finales del producto.

Además de la arquitectura, se seleccionaron una serie de frameworks que apoyaran la arquitectura de referencia:

- Para el manejo de registros de auditoria se definió Log4j, por dos motivos fundamentales, uno es que es el framework más conocido y utilizado en aplicaciones java para el manejo de logs y el otro es su libre distribución.

- Para el manejo de reportes se utilizara JasperReports, al igual que el framework anterior es de libre licenciamiento.

También, se definieron los siguientes diagramas que la fábrica recibirá para la generación de la aplicación:

- El diagrama de clases porque es el que define las clases y características que tiene la aplicación. Este diagrama deberá ser exportado en formato XML para que la fábrica lo pueda interpretar.

- El diagrama entidad-relación ya que provee todo el detalle del modelo de datos necesario para crear el esquema de la base de datos, además, define los objetos de negocio y los DAO de la aplicación. Al igual que el anterior diagrama, también deberá ser exportado en formato XML.

- Un prototipo de vista para la administración de tablas maestras. Este prototipo, deberá ser generado en HTML y ser lo suficientemente genérico para que aplique para cualquier tabla maestra creada en la aplicación.

PASO 5: Seleccionar un generador de código

Para este caso, se eligió velocity como herramienta de generación de código por su facilidad de uso y su licenciamiento gratis, además se comunica con java.

PASO 6: Creación de las plantillas de la fábrica

De acuerdo a los diagramas seleccionados anteriormente y a la arquitectura de referencia definida, se definieron las siguientes plantillas:

- Plantilla de vista para las tablas maestras:
  Es un código HTML combinado con código VTL (Código velocity), en la cual recibiendo los atributos o columnas de la tabla, permite el ingreso, modificación, eliminación y consulta de los datos de dicha tabla.

- Plantilla IDAO:
  Esta plantilla genera una interfaz por cada DAO que sea generado por la fábrica, y tiene predeterminado cinco métodos básicos que se realizan sobre una base de datos que son la conexión a la base de datos y las funciones básicas sobre una tabla (Create, Read, Update, Delete).

- Plantilla DAO:
  Esta plantilla genera la implementación automática de los cinco métodos mencionados anteriormente, además deja abierta la posibilidad de agregar manualmente los métodos específicos del negocio.

- Plantilla IController:
  Esta plantilla tiene los métodos necesarios para delegar las responsabilidades o acciones que son expuestas en las vistas, además se encarga de manejar los mensajes tanto de error como de éxito que se generan desde los objetos del negocio y que se muestran en las vistas.

- Plantilla Controller:
  Esta plantilla tiene la implementación de los métodos anteriormente especificados.

- Plantilla del objeto del negocio:
  Esta plantilla plasma los atributos y funcionalidades que tienen los objetos del negocio definidos en el diagrama de clases.

Estas plantillas, acoplan los diagramas con el generador de código, el cual interpretará los diagramas en formato XML, seleccionara cual es la plantilla a utilizar y generará el código fuente definido tanto en la plantilla como en el diagrama en cuestión.

6. PROCESO DE DESARROLLO SOFTWARE DESPUÉS DE LA IMPLANTACIÓN DE LA FÁBRICA DE SOFTWARE

La introducción de una fábrica de software en un ambiente de desarrollo de software implica cambios grandes en la metodología que se utilizaba antes de esta, ya que aunque si bien la fábrica no genera la totalidad del código fuente, y la etapa de construcción no desaparece, si se modifican las actividades, tareas y tiempos de empleo para cada una de las etapas del ciclo de vida del proceso de desarrollo de software.

Aunque el proceso de desarrollo de software luego de la implantación de una fábrica de software aun no está bien definido ni estandarizado, es posible especificar una adaptación de la metodología tradicional, en donde las etapas del ciclo de vida aumentaran en cantidad, pero disminuirán su complejidad modificando los tiempos estimados para cada una de estas.
A continuación se propondrá una serie de etapas a realizar para la construcción de aplicaciones bajo el apoyo de una fábrica de software:

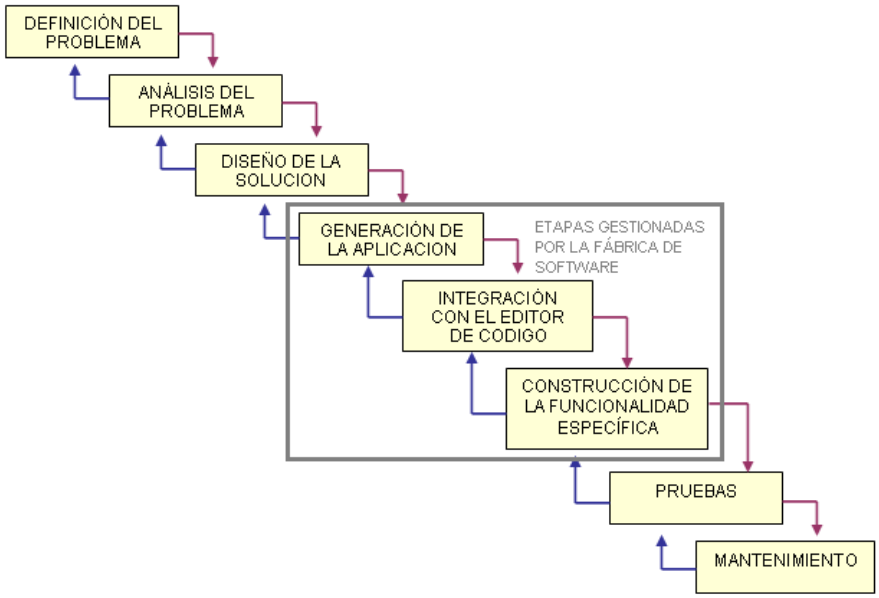


Ilustración 19: Ciclo de vida implementado la fábrica de software

6.1. DEFINICIÓN DEL PROBLEMA

Todo proyecto de construcción de software inicia con esta etapa, en la que se identifica claramente cuál es la necesidad que se tiene, cuál es el problema especifico y los problemas derivados que se quieren solucionar con la implementación de la aplicación, además en esta etapa se analizan los beneficios y consecuencias que traerá dicha implementación.

6.2. ANÁLISIS DEL PROBLEMA

Aquí se define si es viable o no la construcción de la aplicación, se analiza su impacto y se determina cual será la solución a implementar, detallando módulos, funcionalidades y procesos de ejecución de la aplicación. De todas las actividades anteriores, surgen entregables como el diagrama de requisitos y el diagrama de casos de uso.

Entre las actividades a realizar en esta etapa, se debe analizar y definir si la solución deseada o gran parte de ella es alcanzable mediante el uso de la fábrica de software que se tiene implementada.

Si el entorno de desarrollo en el cual se aplicará la fábrica de software es muy maduro, quiere decir que todo proyecto es diseñado con un alto nivel de calidad, lo que implica que sería muy conveniente el uso de la fábrica de software, ya que por un lado la etapa de diseño no aumentará en su tiempo de ejecución y la etapa de construcción si reflejará una disminución notable en su tiempo de ejecución lo que implica menor tiempo de desarrollo total de la aplicación y una reducción notable en costos. Por el contrarío, si la etapa de diseño de la solución no es o era muy importante, quiere decir que los diseñadores de las aplicaciones deben gastar más tiempo en dicha etapa para evitar problemas con su diseño y que se puedan convertir en problemas de generación de la fábrica de software, lo que implicaría que el tiempo ahorrado en la etapa de construcción se gastará en la etapa de diseño y no se verían reflejadas las ventajas del uso de la fábrica de software.

6.3. DISEÑO DE LA SOLUCIÓN

En esta actividad se diseña la aplicación final, se crean los entregables necesarios (materia prima) para la fábrica de software, estos entregables por lo general son los diagramas UML que describen la aplicación. En la mayoría de los casos, estos diagramas son los de clases y el modelo entidad – Relación.

Si la implementación de la solución del problema implica el uso de la fábrica de software, esta etapa se convierte en crítica, ya que los diagramas creados deben reflejar claramente la solución deseada, estar acordes a la arquitectura de referencia establecida para la fábrica y no deben haber errores en dichos diagramas porque implicaría un mal funcionamiento del código generado por la fábrica y por ende se entraría en un reproceso para identificar y solucionar el problema.

6.4. GENERACIÓN DE LA APLICACIÓN

En esta etapa es donde entra en funcionamiento la fábrica de software, a la cual se le entrega como materia prima los diagramas creados en las etapas anteriores. La fábrica, crea el código fuente descrito por la Metadata de los diagramas que recibe como insumos y asegura la flexibilidad, escalabilidad, seguridad, interoperabilidad, entre otras características no funcionales con las cuales fue creada la fábrica de software.

6.5. INTEGRACIÓN CON EDITOR DE CÓDIGO UTILIZADO

Esta etapa es muy simple, consta de integrar el código fuente generado en la actividad anterior con la herramienta o editor de código utilizado (IDE), esta integración se puede hacer a través de la misma fábrica generando los archivos necesarios de descripción de proyectos o creando manualmente los proyectos en la herramienta y agregando el código generado.

6.6. CONSTRUCCIÓN DE LA FUNCIONALIDAD ESPECÍFICA

Aunque una familia de productos generada a través de una fábrica de software tiene muchas funcionalidades en común, se deben programar manualmente las funcionalidades específicas que no las suple la fábrica de software y que son únicas para cada aplicación que se genera.

Esta etapa es la que mas impacta la utilización de la fábrica de software, ya que la estructura y funcionalidades básicas de la aplicación ya están creadas y basta únicamente con agregar el código fuente que describe las situaciones y necesidades especificas del negocio. Si se realizó un muy buen diseño tanto de la fábrica de software como de la aplicación a generar en ella, esta etapa termina siendo en la que más tiempo de ejecución se reduce.

6.7. PRUEBAS

Esta es otra de las etapas más impactadas con el uso de la fábrica de software, ya que solo se prueban las funcionalidades especificas del negocio, debido a que los requisitos no funcionales que normalmente se prueban como seguridad, escalabilidad, portabilidad, entre otros, deben ser soportados por la arquitectura de referencia de la fábrica y ya han sido probados y certificados, lo que hace que esta etapa igual que la anterior, reduzca notablemente su tiempo de ejecución.

6.8. MANTENIMIENTO

Esta etapa se divide en dos actividades muy importantes y son, el mantenimiento de la aplicación generada y el mantenimiento del código generado por la fábrica. El mantenimiento de la aplicación generada solo se debe hacer sobre la funcionalidad específica creada por los desarrolladores, esta tarea se debe realizar manualmente.

Por otro lado, el mantenimiento de la generación de la fábrica de software, se da por motivos funcionales como lo son la modificación de la arquitectura de referencia y la solución de bugs de generación, y por motivos no funcionales como por ejemplo el rendimiento de las aplicaciones generadas. Cuando se modifica la estructura de las funcionalidades que genera una fábrica de software, no implica que las aplicaciones que ya se generaron no puedan ser actualizadas, ya que si la fábrica de software fue bien definida, puede regenerar las

aplicaciones sin dañar las funcionalidades ya introducidas por los desarrolladores.

7. FUTURAS INVESTIGACIONES

Los temas que se mencionan en este documento como arquitectura de referencia, entre otros, permiten para futuras investigaciones profundizar más sobre ellos, ya que se definen de manera general, para no quitarle importancia al tema central y no extenderse o desviarnos del alcance.
Los temas en los cuales se puede profundizar en las investigaciones posteriores son:

- Pasos para la definición de la arquitectura de referencia
- Otros tipos de patrones existentes en los temas transversales al software
- Temas transversales al software como gestión de la configuración, administración de los activos, toma de decisiones entre otras.

Adicional a esto, y teniendo en cuenta el tema principal del documento, se puede implementar una fábrica de software teniendo en cuenta los pasos propuestos en esta guía.

8. CONCLUSIONES

- Los pasos propuestos en la guía muestran las etapas elementales que se deben seguir para crear una fábrica de software simple y fácil de implementar, ya que provee información básica y necesaria para su creación.

- Seguir los pasos propuestos en la guía ayuda no solo a la ejecución de un proyecto de creación de una fábrica de software sino a su gestión, ya que posibilita la identificación de las actividades, cronogramas, el control y el manejo de los recursos a utilizar.

- La guía metodológica propuesta en el documento está enfocada en el desarrollo de software, por lo cual hay que tener presente otros aspectos transversales al proceso de desarrollo como la gestión del producto generado por la fábrica, administración de activos de software, gestión de la configuración de la fábrica de software y de los productos generados por esta, entre otras.

- Para llevar a cabo el desarrollo de una fábrica de software bien elaborada se requiere de un gran conocimiento en los temas que son base para la construcción de la misma como arquitectura de referencia, patrones, frameworks, entre otros que permiten de alguna u otra manera facilitar el proceso de desarrollo de software.

- En la actualidad las industrias crean productos de forma masiva, lo cual les permite satisfacer en el menor tiempo posible la demanda del mismo. En el campo del desarrollo de software falta madurez en las técnicas y metodologías que se deben implementar para que pueda ocurrir, sin embargo se espera que las fábricas de software sean una contribución importante para lograr este objetivo.

- Las fábricas de software ayudan a optimizar y minimizar los tiempos y esfuerzos en el proceso de desarrollo de una aplicación software, permitiendo a los analistas enfocarse en las etapas de análisis y diseño de las aplicaciones y en la lógica propia del negocio.

- El aprovechamiento de la reutilización en las fábricas de software y el pensamiento de crear productos software de forma masiva, permite adaptar la industria del software a las economías de escala y de alcance.

- Con la implementación de una fábrica de software, algunos de los entregables que se generan en el análisis y el diseño de una aplicación, dejarán de ser activos de desarrollo para los programadores, y se convertirán en materia prima para las fábricas de software, haciendo que todo el esfuerzo y dedicación que se realiza en dichas etapas del desarrollo de software no solo sean explotadas a nivel conceptual sino que además permitan disminuir notoriamente el esfuerzo requerido en la etapa de construcción.

10. BIBLIOGRAFÍA

Libros:

- Piattini Velthuis, Mario / Garzas Parra, Javier – Fábricas de software: experiencias, tecnologías y organización – Ra-ma.

- Larman, Craig – UML y Patrones: Introducción y Diseño Orientado a Objetos – Prentice Hall.

- Gamma, Erich / Helm, Richard / Johnson, Ralph / Vlissides, John – Design Patterns Elements of Reusable Object Oriented Software – Addison wesley.

- Pressman, Roger – Software Engineering A Practitioner's Approach – McGraw-Hill.

Referencias Web:

- Greenfield, Jack – Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.
  http://msdn2.microsoft.com/en-us/library/ms954811.aspx

- Greenfield, Jack / Short, Keith / Cook, Steve / Kent, Stuart – Software factories, industrialized software development.
  http://www.softwarefactories.com

- Souto, Marcos Abel – Patrones de software: Breve introducción.
  http://www.elrincondelprogramador.com/default.asp?pag=articulos/leer.asp&id=12

- Patrón de diseño.
  http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o

- Molpeceres, Alberto – Diseño de software con patrones.

http://www.programacion.com/java/articulo/joa_patrones1/

- Hurtado Jara, Omar – Evolución y orientaciones de patrones. http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml#tipos

- Framework. http://es.wikipedia.org/wiki/Framework

- Qué es un framework de desarrollo web. http://euphoriait.com/articulos/framework_web

- Software framework. http://en.wikipedia.org/wiki/Software_framework

- Markiewicz, Marcus Eduardo / Lucena, Carlos – Object oriented framework development. http://www.acm.org/crossroads/xrds7-4/frameworks.html

- Clifton, Mark – What is a framework? http://www.codeproject.com/KB/architecture/WhatIsAFramework.aspx

- Davis, Malcolm – Struts, an open-source MVC implementation. www.ibm.com/developerworks/web/library/j-struts/

- Rotem, Arnon – Frameworks vs Libraries. http://www.ddj.com/blog/architectblog/archives/2006/07/frameworks_vs_l.html

- Arquitectura de referencia SOA. http://soaagenda.com/journal/articulos/arquitectura-de-referencia-soa/

- Reference architecture. http://en.wikipedia.org/wiki/Reference_architecture

- The importance of reference architecture.

http://www.architectureandchange.com/2007/12/29/the-importance-of-reference-architecture/

- Krueger, Charles – Software product lines. www.softwareproductlines.com

- Smart client software factory. http://msdn.microsoft.com/en-us/library/aa480482.aspx

- Web cliente software factory. http://www.codeplex.com/websf

- The Apache velocity Project. http://velocity.apache.org/engine/devel/translations/user-guide_es.html

- MyGeneration code generador, O/R mapping, and architectures. www.mygenerationsoftware.com

- Acceleo generador de código fuente. http://www.acceleo.org/wiki/index.php/Main_Page

- Metodología de análisis y eliminación de los problemas. http://www.monografias.com/trabajos22/solucion-de-problemas/solucion-de-problemas.shtml

Norma de elaboración de documentos:

- ICONTEC, Norma Técnica Colombiana NTC-ISO 9001:2000, Sistema de Gestión de la Calidad.

- Universidad EAFIT – Reglamento de proyectos de grado de la escuela de ingeniería. http://www.eafit.edu.co/EafitCn/Institucional/Reglamento/RProyectosIngenieria

APENDICE A

Historia de los patrones de software

La primera vez que se escucha la palabra patrón es en el año de 1979, cuando el arquitecto Christopher Alexander escribió un libro titulado “The Timeless Way of Building”, donde proponía un aprendizaje y uso de los patrones que allí había escrito, para la construcción de edificios con una mejor calidad.
Alexander definió la palabra patrón así: “… Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez.”. En años posteriores Alexander con una serie de colegas publicaron un volumen denominado “A Pattern Language” siendo este un intento de formalizar un conjunto de soluciones, y formalizar y plasmar años y años de conocimiento arquitectónico. En este volumen los participes definieron un patrón como “… Un patrón define una posible solución correcta para un problema de diseño dentro de un contexto dado, describiendo las cualidades invariantes de todas las soluciones.”

Más tarde en los años 80’s, Ward Cunninghan y Kent Beck se basaron en las ideas del arquitecto Alexander para crear cinco patrones que permitían la interacción hombre – máquina y publicaron un articulo con el nombre “Using Pattern Languages for OO Programs” en el año de 1987.

En los años 90’s cuando los patrones comenzaron a tomar fuerza en el mundo de la informática, se publico un libro con el nombre “Design Patterns” escrito por el GoF (Gang of Four), donde consignaron 23 patrones de interés y de gran uso en el desarrollo de aplicaciones software.

APENDICE B

Herramientas de generación y construcción de fábricas de software

Velocity

Velocity es un motor de plantillas basado en Java, se puede utilizar para crear páginas Web, SQL, PostScript y cualquier otro tipo de salida de plantillas. Se puede utilizar como una aplicación independiente para generar código fuente y reportes, o como un componente integrado en otros sistemas.

Velocity provee un lenguaje de plantillas denominado VTL, el cual fue creado para proveer de una manera fácil la posibilidad de incluir contenido dinámico dentro de archivos. VTL usa referencias para incluir el contenido dinámico, una variable es un tipo de referencia que puede referirse a un objeto definido dentro del código Java u obtener su valor de un enunciado VTL dentro del mismo archivo.

Con velocity, se pueden definir las plantillas que implementan la arquitectura de la fábrica de software a crear, para luego obtener dinámicamente los valores necesarios del los archivos XML o XMI para generar la aplicación.

El siguiente link referencia la pagina principal de apache velocity en español: http://velocity.apache.org/engine/devel/translations/user-guide_es.html

Mygeneration

MyGeneration al contrario de Velocity, es una herramienta desarrollada en Microsoft .NET, genera código de plantillas que pueden ser escritas en C#, Visual Basic .NET, JScript y VBScript, además, provee la posibilidad de generar código fuente para sistemas operativos que no sean de Microsoft. Permite generar a través de plantillas, código fuente de los lenguajes de programación C# y Visual Basic .NET, genera procedimientos almacenados, código PHP, HTML entre otros.

Acceleo

Acceleo es otra de las herramientas gratuitas que existen en el mercado para la generación de código fuente.

El código fuente que se genera por esta herramienta es a partir de modelos existentes.

Esta es una herramienta diseñada para las personas que están dispuestas a sacar provecho en aumentar la productividad en el desarrollo de software.

Esta herramienta permite la generación de archivos o código fuente a través de UML, MOF, EMF y módulos.

Las características de Acceleo son:

- Completa integración con la herramienta Eclipse y el framework EMF.
- Sincronización entre el código y el modelo.
- Generación incremental
- Facilidad de adaptación de los proyectos organizacionales.
- Facilidad de actualización y mantenimiento de plantillas.
- Auto-completación y detección de errores.

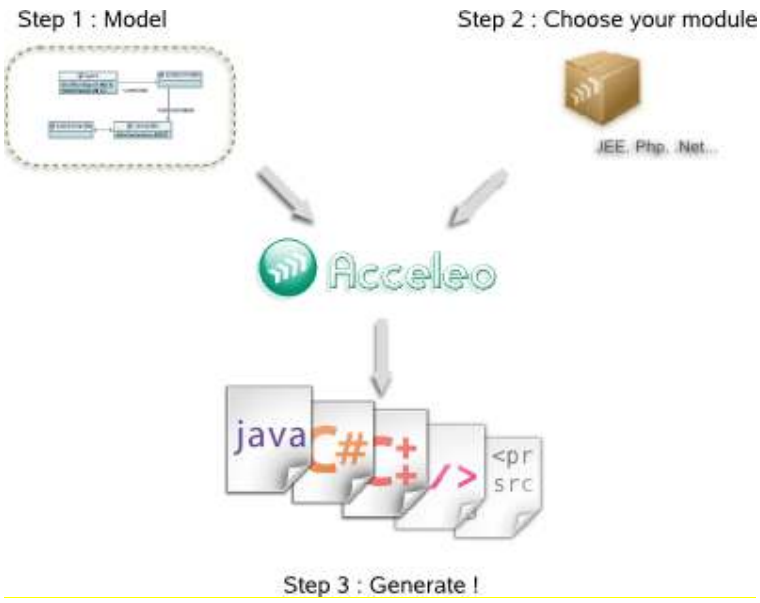


Ilustración 20: Esquema de generación de código de Acceleo

Lenguajes de programación

Otra opción que es muy acertada es la de desarrollar el software propio para creación de la línea de producción de la fábrica de software. Los lenguajes de programación actuales proveen funcionalidades de lectura de archivos XML y XMl que son la materia prima de la fábrica, además posibilitan la creación de archivos de cualquier tipo. Además el tiempo que se puede gastar en conocer una herramienta externa, se puede emplear en perfeccionar el diseño y la implementación de la fábrica.