



COMPONENTE ARQUITECTÓNICO PARA LA GENERACIÓN DE REPORTE

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR

DAYANA DANIEL HERNANDEZ

TUTOR

ING. YADIEL RAMOS RODRIGUEZ

Ciudad de La Habana, 2009

“Año del 50 Aniversario del Triunfo de la Revolución”

Declaración de Autoría

Declaro que soy el autor de este trabajo y autorizo a la Facultad 8 de la Universidad de las Ciencias Informáticas; así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2009.

Dayana Daniel Hernández

Ing. Yadiel Ramos Rodríguez

Firma del Autor

Firma del Tutor

AGRADECIMIENTOS

A la Revolución, a Fidel y a la UCI por hacer posibles mis sueños.

A mis padres, por el amor y la confianza que siempre me han dado. Por ser la guía en mi camino. Por ser la razón de mi empeño. Por estar siempre para mí.

A mi novio, por estar a mi lado todo este tiempo. Por su esfuerzo cada día en hacer de mí una mejor profesional. Por todos los conocimientos que me ha aportado. Por ser mi escuela como ingeniera.

A mi hermano, mis abuelos y mi tía, por todo el cariño y apoyo que nunca me faltó.

A mi familia, por su confianza.

A mi tutor, por todas sus ideas y críticas, que me ayudaron a la construcción de este trabajo.

DEDICATORIA

A mi familia y a Yadiel, que son la razón de mi existencia y de mi esfuerzo.

A mi amiga Danay, que aunque la vida no se lo permitió, sé que le habría gustado estar aquí conmigo; para ella en especial es este trabajo.

“La función de un buen software es hacer que lo complejo aparente ser simple”

Grady Booch

RESUMEN

El presente trabajo tiene como objetivo diseñar e implementar un componente arquitectónico para la generación de reportes en el proyecto CICPC de la Universidad de Ciencias Informáticas. Para ello se utilizó la librería JasperReports por las facilidades que brinda, así como todas aquellas de las que depende para su correcto funcionamiento. Se seleccionó la metodología XP como rectora del proceso de desarrollo obteniéndose un componente en los espacios de tiempo y coste esperados, que cumple con las características requeridas por los clientes y da solución a los problemas existentes dentro del proyecto con relación a la generación de reportes.

ABSTRACT

This work aims to design and implement an architectonic component for generating reports on the project CICPC of the University of Informatics Science. This involved the use of facilities provides by the JasperReports library, as well as those on which it depends for its proper functions. XP methodology was selected as the leadership development process obtaining a component in the expected space of time and cost, which complies with the characteristics, required by customers and provides solutions to existing problems within the project in relation to the reports generation.

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN	1
CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA	5
1.1. ARQUITECTURA DE SOFTWARE	5
1.2. HERRAMIENTAS DE DESARROLLO	8
Crystal Reports para Java	9
Birt (Business Intelligence and Reporting Tools)	9
JasperReports	10
Selección de la Herramienta	13
JFreeChart	14
iReport	15
Librerías	16
1.3. METODOLOGÍAS DE DESARROLLO	17
RUP	18
FDD	19
XP	20
Selección de la Metodología	21
1.4. CONCLUSIONES.....	23
CAPÍTULO 2. PROPUESTA DE SOLUCIÓN	24
2.1. EXPLORACIÓN.....	25
Ciclo de Vida de JasperReports	26
Historias de Usuario	31
2.2. PLANEACIÓN	32
2.3. CONCLUSIONES.....	42
CAPÍTULO 3. IMPLEMENTACIÓN DE LA SOLUCIÓN	43

3.1.	ITERACIONES PARA LA PRODUCCIÓN DE ENTREGABLES	43
	Tareas de Implementación	43
	Diseño	43
	Implementación	45
	Pruebas de Unidad	60
3.2.	PRODUCTIZACIÓN	61
3.3.	CONCLUSIONES.....	61
CAPÍTULO 4.	VALIDACIÓN DE LA SOLUCIÓN	62
4.1.	MANTENIMIENTO.....	62
4.2.	VISIÓN GLOBAL DEL TRABAJO REALIZADO	64
	Ahorro en Tiempo de Desarrollo	64
	Ahorro de Líneas de Código	66
4.3.	ANÁLISIS DE FACTIBILIDAD	66
	Pruebas Realizadas	68
4.4.	ASPECTOS FAVORABLES.....	72
	Utilización de Soluciones Recientes e Innovadoras	72
	Soluciones a Problemas Detectados en las Librerías	74
	Nueva Documentación de la Librería	75
4.5.	ASPECTOS DESFAVORABLES.....	76
4.6.	CONCLUSIONES.....	77
CONCLUSIONES	78	
RECOMENDACIONES	79	
REFERENCIAS BIBLIOGRÁFICAS	81	
BIBLIOGRAFÍA	84	
GLOSARIO DE TÉRMINOS	86	
ANEXOS	88	
	ANEXO 1: PLANTILLAS PARA REPORTES DE CICPC	88

ANEXO 2: TAREAS DE IMPLEMENTACIÓN. PRIMERA ITERACIÓN	90
ANEXO 3: TAREAS DE IMPLEMENTACIÓN. SEGUNDA ITERACIÓN	92

ÍNDICE DE FIGURAS

FIGURA 1.1: PATRÓN ARQUITECTÓNICO DE 3 CAPAS SELECCIONADO PARA SER USADO EN EL PROYECTO CICPC Y REPRESENTACIÓN DE LOS FRAMEWORKS DE DESARROLLO DENTRO DEL MISMO.	8
FIGURA 1.2: ARQUITECTURA DE JASPERREPORTS.....	13
FIGURA 1.3: GRÁFICO DE PASTEL CREADO CON LA LIBRERÍA JFREECHART	14
FIGURA 1.4: GRÁFICO DE BARRAS CREADO CON LA LIBRERÍA JFREECHART.....	14
FIGURA 1.5: GRÁFICO DE LÍNEAS CREADO CON LA LIBRERÍA JFREECHART	15
FIGURA 1.6: GRÁFICO DE PILA CREADO CON LA LIBRERÍA JFREECHART	15
FIGURA 1.7: FASES Y FLUJOS DE TRABAJO EN RUP	18
FIGURA 1.8: PROCESO FDD	20
FIGURA 1.9: CICLO DE VIDA DE XP	21
FIGURA 2.1: PROPUESTA DE SOLUCIÓN.....	25
FIGURA 2.2: CICLO DE VIDA DEL REPORTE	26
FIGURA 2.3: ARCHIVO .JRXML.....	27
FIGURA 2.4: DISEÑO DE CLASES QUE COMPONEN EL OBJETO JASPERDESIGN	29
FIGURA 2.5: PROCESO SEGUIDO EN LA FASE DE PLANEACIÓN.....	32
FIGURA 2.6: PRIMERA ITERACIÓN.	40

FIGURA 2.7: SEGUNDA ITERACIÓN	41
FIGURA 3.1: DISTRIBUCIÓN DE PAQUETES EN EL COMPONENTE	44
FIGURA 3.2: TARJETAS CRC.....	46
FIGURA 3.3: ESTRUCTURA DE LAS TABLAS EN JASPERREPORTS.....	49
FIGURA 3.4: ESTRUCTURA DE LOS GRÁFICOS EN JASPERREPORTS	51
FIGURA 3.5: CICLO DE VIDA DE JASPERREPORTS INTERCEPTADO POR EL COMPONENTE DE REPORTES	58
FIGURA 4.1 : MANEJO DE LOS REPORTES EN CICPC	67
FIGURA 4.2: PROBLEMAS DETECTADOS DURANTE LAS PRUEBAS RELACIONADOS CON LA GENERACIÓN DE REPORTES.....	69
FIGURA 4.3: PROBLEMAS DETECTADOS A LA APLICACIÓN	70
FIGURA 4.4: PROBLEMAS DETECTADOS DURANTE LAS PRUEBAS RELACIONADOS CON LA GENERACIÓN DE REPORTES POR EL USUARIO FINAL.....	71
FIGURA 4.5: DESCARGAS DE JASPERREPORTS-3.1.0 EN EL ÚLTIMO PERÍODO.....	72
FIGURA 4.6: DESCARGAS DE IREPORT-1.3.3 EN EL ÚLTIMO PERÍODO	73
FIGURA 4.7: DESCARGAS DE JFREECHART-1.0.12 EN EL ÚLTIMO PERÍODO	73

INTRODUCCIÓN

Toda aplicación de gestión cumple un principio básico: Todas tienen datos, conectividad sobre éstos y forma de visualizar, analizar, capturar y procesar los mismos para los usuarios del sistema.

Para la tarea de procesar y mostrar datos entran en juego los Reportes que no son más que objetos que entregan información en un formato particular y que permiten realizar ciertas operaciones como imprimirlos, enviarlos por email, guardarlos a un archivo, entre otras.

Entre los proyectos productivos de la Universidad de las Ciencias Informáticas se encuentra el proyecto CICPC, que consiste en automatizar el Cuerpo de Investigaciones Científicas Penales y Criminalísticas de Venezuela y que arrojará como resultado final una aplicación de gestión que deberá adaptarse al principio básico mencionado anteriormente.

Actualmente en el proyecto CICPC no existe un componente que, adaptándose a la arquitectura a utilizar, facilite la interacción entre los programadores y los procesos necesarios para la generación de reportes en java. Esto trae consigo que se relentice el proceso de desarrollo al dificultarse las tareas y hacerse necesario la repetición de código en varios lugares del software para darle solución a los requisitos relacionados con reportes. Además de requerir por parte del programador la escritura de código de bajo nivel, haciéndose nula la reutilización de componentes de software.

Por lo cual el **problema de investigación** está dado por la necesidad de construir un componente que encierre la lógica asociada a la generación de reportes del proyecto CICPC.

El problema anterior enmarca la investigación en los procesos para la generación de reportes como **objeto de estudio**, más específicamente en los procesos para la generación de reportes en java que se ajusten a la arquitectura de CICPC como **campo de acción**.

El **objetivo general** propuesto es desarrollar un componente arquitectónico que encierre la lógica asociada a la generación de reportes en el proyecto CICPC. Desglosando el mismo se obtienen los siguientes **objetivos específicos**:

- Valorar el estado de la generación de reportes en java a partir de un estudio de la información referente a los mismos.
- Definir las funcionalidades referentes a la generación de reportes que debe brindar el software a construir.

- Diseñar el componente propuesto.
- Implementar la infraestructura necesaria para el componente propuesto.
- Comprobar la validez de la solución final.

Teniendo en cuenta lo anterior la **idea a defender** sería la siguiente: La creación de un componente arquitectónico para la generación de reportes en el proyecto CICPC acelerará el proceso de desarrollo y arrojará un producto menos expuesto a errores.

Para dar cumplimiento a los objetivos propuestos se plantean una serie de tareas relacionadas a continuación:

Tareas Investigativas

Tareas generales de investigación

1. Analizar los logros y limitaciones en los enfoques existentes sobre los reportes y el papel que juegan en una aplicación de gestión.
2. Estudiar el concepto de componente arquitectónico y su funcionalidad en el desarrollo de software.
3. Analizar la arquitectura propuesta para el proyecto CICPC y el impacto que tendrá la misma sobre el componente.
4. Evaluar las herramientas existentes con funcionalidades semejantes a las que se requieren y la posibilidad de reutilizarlas en el componente a desarrollar.
5. Analizar y clasificar las funcionalidades necesarias y deseables para ser implementadas en la solución final.

Tareas del proceso de ingeniería y desarrollo del software

1. Definir el proceso de desarrollo a seguir dentro de la filosofía de XP adaptándolo a las características particulares del trabajo a desarrollar.

Tareas de análisis e implementación del producto

1. Identificar las historias de usuario a implementar en cada iteración.

2. Estudiar la relación existente entre las librerías de desarrollo seleccionadas y los frameworks presentes en la aplicación.
3. Implementar pruebas de unidad.
4. Diseñar el componente.
5. Implementar las historias de usuario identificadas.
6. Desarrollar pruebas funcionales con el cliente.
7. Realizar estudios sobre la validez de la solución final.

El presente trabajo está dividido en cuatro capítulos. El primero de ellos se ha denominado *Fundamentación Teórica* porque en él se analiza el concepto de reporte en cuestión; el enfoque arquitectónico que deben tener los componentes de un software; las herramientas existentes a nivel internacional y el aporte que puedan brindar durante la implementación; las metodologías de desarrollo del software y su papel primordial en el logro de un producto con la calidad requerida y en el tiempo y costo planificados; para con estos conocimientos sintetizar en cómo se podrá construir un componente arquitectónico para la generación de reportes que apoye el desarrollo de una aplicación de gestión utilizando de forma óptima las herramientas existentes a nivel internacional y cumpliendo con la calidad requerida y en el tiempo y costo planificados.

El capítulo dos, titulado *Propuesta de Solución*, define las funcionalidades necesarias a incluir en la solución que se propone así como la estrategia a seguir para la implementación de la misma adecuándose a los plazos de tiempo impuestos por los usuarios. Además resume los resultados del estudio de las herramientas para la generación de reportes a utilizar y provee de los artefactos obtenidos a raíz de la aplicación de la metodología de desarrollo seleccionada.

El tercer capítulo, nombrado *Implementación de la Solución*, expone el proceso seguido para la construcción del componente arquitectónico haciendo énfasis en sus características esenciales y la forma de dar cumplimiento a las funcionalidades identificadas, así como las restricciones surgidas para garantizar el buen funcionamiento del mismo. Además provee de un conjunto de artefactos obtenidos a raíz de la aplicación de la metodología de desarrollo seleccionada.

En el cuarto y último capítulo, denominado *Validación de la Solución*, se verifican de manera unitaria e integrada todos los elementos del componente de software y se realiza junto a los usuarios finales la

validación de la solución desarrollada a fin de contar con todas las evidencias que garanticen el cumplimiento en alcance, funcionalidad y calidad que el cliente espera. Además se realiza un estudio de los aspectos favorables y desfavorables que tuvieron lugar durante el desarrollo de la solución y el impacto social de la misma haciendo énfasis en los aportes alcanzados sobre los procesos para la generación de reportes y las herramientas empleadas para llevarlos a cabo.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Una aplicación de gestión se caracteriza por estar formada en gran escala por reportes. Son estos los encargados de dar forma a los datos involucrados y presentarlos al usuario como documentos formales y organizados. La generalidad de los reportes, por lo tanto, permitirá que sean analizadas las técnicas y herramientas existentes para la construcción de los mismos.

En el presente capítulo se analizará el concepto de reporte en cuestión; el enfoque arquitectónico que deben tener los componentes de un software; las herramientas existentes a nivel internacional y el aporte que puedan brindar durante la implementación; las metodologías de desarrollo del software y su papel primordial en el logro de un producto con la calidad requerida y en el tiempo y costo planificados.

Después de obtener todos estos conocimientos se sintetizará en cómo se podrá construir un componente arquitectónico para la generación de reportes que apoye el desarrollo de una aplicación de gestión utilizando de forma óptima las herramientas existentes a nivel internacional y cumpliendo con la calidad requerida y en el tiempo y costo planificados.

1.1. ARQUITECTURA DE SOFTWARE

Un componente arquitectónico representa un conjunto de clases con propiedades similares y con un fin común que interactúan con otros componentes o subsistemas mediante interfaces. Los mismos están ligados a la arquitectura del software y por lo tanto no se pueden construir como partes separadas, sino como complementos de un todo. Por lo tanto, es necesario analizar en primer lugar la arquitectura candidata propuesta para el proyecto CICPC y de la que el componente en cuestión formará parte.

Una de las tareas fundamentales que se trazó el equipo de desarrollo del proyecto una vez conformado, fue construir una arquitectura flexible, robusta y adaptada a precondiciones impuestas por los clientes. Estas precondiciones recogen la necesidad de una aplicación web única y de alcance nacional, el uso de la tecnología Java, el uso de Oracle como gestor de datos, el uso de AFIS (Sistema Automático de Identificación de Huellas Dactilares), la ejecución del software en un ambiente controlado (VLAN), la ausencia de acceso físico al servidor de aplicaciones y de datos sin utilizar encriptación de archivos de configuración, el uso de autenticación por huellas digitales, un sistema exhaustivo de auditorías y una aplicación cross browser compatible para las versiones 6 o mayor del Internet Explorer y para el Mozilla Firefox. (1)

Ante las mismas se realizó un estudio de las tecnologías y herramientas compatibles y que facilitarían su cumplimiento, el cual arrojó los siguientes resultados:

Como herramientas para el desarrollo se seleccionaron en la plataforma java la JDK 1.6, la cual proporciona un acceso más rápido a arreglos críticos, un plan de soporte completo, y funcionalidades diseñadas para reducir el costo de desarrollo. (2)

Como contenedor web el Tomcat 6.0, el cual funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems. Dado que fue escrito en Java, funciona en cualquier sistema operativo que disponga de una máquina virtual Java. Es cada vez más utilizado por las empresas en los entornos de producción debido a su contrastada estabilidad. Entre los avances que incluye la versión utilizada se encuentran que cuenta con implementaciones para Servlet 2.5 y JSP 2.1, soporte para Unified Expression Language 2.1 y que fue diseñado para funcionar en Java SE 5.0 y posteriores. (3)

Para el control de versiones se seleccionó el subversion, el cual posee las siguientes características: versionado de directorios, versionado de cambios sobre archivos, atomicidad, metadatos, modos de acceso, diff binarios, branches y tags. (4)

Como herramienta de modelado visual paradigm for UML 6.4, el cual es una herramienta de modelado UML profesional que soporta el ciclo completo de desarrollo del software, el análisis y diseño orientado a objetos, la construcción, las pruebas y el despliegue. El modelado UML permite construir aplicaciones de alta calidad más rápido, mejores y a un menor costo. Se pueden crear toda clase de diagramas, realizar ingeniería inversa, generar código desde los diagramas y generar documentación. (5)

Para el ambiente de desarrollo se seleccionó RedHatDeveloperStudio que incluye el Eclipse 3.3. Con el mismo se provee a los desarrolladores de herramientas basadas en Eclipse integradas y un entorno disponible completamente open source. La piedra angular del programa para desarrolladores de Red Hat, Developer Studio, está diseñada para maximizar la productividad del desarrollador sobre las soluciones de Red Hat y, en última instancia, acelerar el cambio empresarial a una arquitectura open source. (6)

Developer Studio combina productos aportados a Red Hat por Exadel en marzo de 2007 -Exadel Studio Pro, RichFaces y Ajax4jsf- con software de middleware Jboss como JBoss Seam e Hibernate dentro de un potente entorno de desarrollo para aplicaciones SOA, Ajax y Java empresariales.

Además se incluyeron plugins adicionales de integración con el ambiente como subEclipse 1.4, el cual provee soporte para Subversion en el IDE Eclipse. (7)

Se seleccionó un estilo arquitectónico en capas que proporciona modularidad al proyecto, específicamente de 3 capas (Figura 1.1), las cuales estarían representadas por los frameworks que se citan a continuación:

Hibernate, para la persistencia y acceso a datos, el cual es una capa de persistencia objeto/relacional y un generador de sentencias SQL. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada se puede generar bases de datos en cualquiera de los entornos soportados: Oracle, DB2, MySql, entre otras. (8)

Spring, para la capa media de lógica de negocio, el cual facilita la creación de componentes reutilizables, además de que se adapta fácilmente con otros frameworks como lo son Hibernate, iBatis y Struts. (9)

JSF (Java Server Faces) para la capa de presentación el cual es un estándar sencillo que aporta los componentes básicos de las páginas web además de permitir crear componentes más complejos como menús, pestañas, árboles, entre otros. (10)

Acegi Security System para las restricciones de seguridad el cual proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java utilizando características de programación orientada a aspectos, de forma transparente para el desarrollador, sin necesidad de desarrollar código, utilizando para ello el soporte prestado por el framework Spring. (11)

JUnit el cual es la herramienta especialmente diseñada para implementar y automatizar la realización de pruebas de unidad en Java. (12)



Figura 1.1: Patrón arquitectónico de 3 capas seleccionado para ser usado en el proyecto CICPC y representación de los frameworks de desarrollo dentro del mismo.

Ante este modelo sólo queda la selección de las herramientas para la generación de reportes compatibles con la arquitectura propuesta y la construcción del componente que formará parte de la misma.

1.2. HERRAMIENTAS DE DESARROLLO

Son muchas las herramientas reconocidas a nivel internacional para la generación de reportes, sin embargo, las restricciones antes mencionadas imponen centrar el estudio en aquellas herramientas escritas en java y de código libre. De ellas, existen tres herramientas líderes en el desarrollo de software, Crystal Report, BIRT y JasperReports.

CRYSTAL REPORTS PARA JAVA

Crystal Reports para Java simplifica y agiliza el proceso de acceso a los datos, formateo e integración en las páginas JavaServer Pages (JSP) y está disponible para utilizarlo directamente desde el entorno integrado de desarrollo (IDE) Eclipse. (13)

Utilizando Crystal Reports para Java dentro de estos entornos integrados de desarrollo (IDE), se podrán obtener reportes que cumplan con los requisitos impuestos por el cliente, facilitando su elaboración y dando cumplimiento a las normas de calidad que los documentos formales requieren.

Sin embargo, Crystal cuenta con una limitada fuente de datos a la hora de generar los reportes impidiendo así el uso efectivo de la información con que se cuenta. Su compatibilidad se limita al sistema operativo Windows, restringiendo así su uso para sistemas operativos de software libre. El formato de salida estándar es el PDF. Además, se ha caracterizado por la falta de documentación, la ausencia de foros especializados, la mala calidad del soporte, la ausencia de información durante el desarrollo (trazas, herramientas de seguimiento, entre otras) y falta de compatibilidad con el resto de las librerías disponibles.

BIRT (BUSINESS INTELLIGENCE AND REPORTING TOOLS)

Birt es un sistema de reportes de código libre basado en eclipse para aplicaciones web, especialmente si están basadas en java y J2EE. Tiene 2 componentes principales, una herramienta para diseñar los reportes basada en eclipse, y un componente que se puede agregar al servidor de la aplicación. Además permite agregar gráficos a la propia aplicación. (14)

Con Birt, se puede incluir gran variedad de reportes en la aplicación, cumpliendo con las funcionalidades requeridas y generando reportes que cumplen con las normas de calidad impuestas. Posee variadas fuentes de datos que permiten al usuario obtener los mismos del lugar más conveniente. Puede usarse en cualquier plataforma donde eclipse corra. Posee variados formatos de salida como HTML, PDF, WORD y XLS.

Birt como proyecto cuenta con un sitio web de preguntas comunes con sus respuestas y una ayuda a los que se inician con la herramienta de los pasos iniciales a seguir.

JASPERREPORTS

JasperReports es la librería más popular de reportes de código libre en java. Se puede incluir fácilmente en cualquier aplicación Java para la creación de sofisticados reportes. Se utiliza además para crear archivos de salida que se emplearán en el futuro, como hojas de cálculo.

Brinda las funcionalidades necesarias para dar cumplimiento a los requisitos impuestos. Además de los datos en forma de texto, JasperReports es capaz de generar reportes profesionales incluyendo imágenes y gráficos, entre sus funcionalidades más importantes se encuentran: (15)

- Provee de un diseño flexible de reportes.
- Es capaz de presentar los datos de forma gráfica o textual.
- Permite a los desarrolladores el suministro de datos de múltiples formas.
- Acepta datos de varias fuentes.
- Permite incluir marcas de agua.
- Puede generar subinformes.
- Es capaz de exportar los reportes a múltiples formatos

A continuación serán brevemente descritas cada una de estas funcionalidades:

Provee de un diseño flexible de reportes.

JasperReports nos permite separar los datos en secciones opcionales dentro del reporte. Dichas secciones incluyen el título del reporte, el encabezado de página, la sección del cuerpo del reporte, el pie de página y el sumario. Además, con él se pueden crear capas dinámicas basadas en el contenido del reporte, por ejemplo, en dependencia del valor de un campo del reporte, los datos pueden ocultarse o mostrarse en el mismo, incluso pueden agruparse en secciones lógicas, por ejemplo, si creamos un reporte de carros, es posible agrupar los datos por modelo, año, o una combinación de estas o cualquier otra pieza de datos mostrada en el reporte. Agrupar datos proporciona un mejor control del diseño del reporte. La definición de datos agrupados puede usarse además para calcular valores subtotales basados en el subconjunto de datos agrupados en el reporte, además pueden ser usados como definiciones para los gráficos.

Es capaz de presentar los datos de forma gráfica o textual.

JasperReports provee la habilidad de mostrar los datos del reporte textualmente o gráficamente haciendo uso de los gráficos. Permite usar expresiones para generar reportes que muestren datos de forma dinámica, lo cual significa que los datos que no son pasados directamente al reporte o almacenados en alguna parte, son calculados a partir de los datos contenidos en el origen de datos o en los parámetros del reporte.

Permite a los desarrolladores el suministro de datos de múltiples formas.

JasperReports permite a los desarrolladores pasar los datos al reporte por parámetros, los cuales pueden ser instancias de cualquier clase de Java o definida en la aplicación. Los datos pueden ser pasados además usando clases especiales llamadas *datasources* (*origen de datos*). Los parámetros y los *datasources* se pueden combinar para lograr una mayor flexibilidad.

Acepta datos de varias fuentes.

JasperReports genera reportes usando cualquier sistema de bases de datos relacional soportado por JDBC. Sin embargo, no está limitado a las bases de datos solamente. Se pueden generar reportes desde un gran número de *datasources*, incluyendo archivos XML, Plain Old Java Objects (POJOs), cualquier clase que implemente la interfaz `java.util.Map` y cualquier clase que implemente la interfaz `javax.swing.TableModel`.

JasperReports soporta además orígenes de datos vacíos, los cuales son usados en reportes simples que no tengan datos dinámicos que mostrar. Además, si se necesita crear un reporte desde un origen de datos que no soporte directamente JasperReports, es posible crear nuestro propio *datasources*.

Permite incluir marcas de agua.

JasperReports es capaz de generar imágenes o texto de fondo en los reportes que genere. Estas imágenes de fondo pueden servir como una especie de marca de agua para los reportes. Una marca de agua es como una imagen secundaria que se establece debajo de las imágenes principales.

Las marcas de agua pueden utilizarse para marcar los reportes o con propósitos de seguridad, pues hacen difícil la posibilidad de falsificar informes. Todas las páginas del reporte tienen la misma marca de agua, la cual les da un aspecto coherente a los mismos.

Puede generar subreportes.

Otra de las funcionalidades de JasperReports es que nos permite crear subreportes, o reportes dentro de reportes. Los subreportes simplifican el diseño del reporte significativamente, permitiendo extraer las secciones complejas de los reportes a reportes separados e incorporar los mismos dentro del reporte maestro.

Es capaz de exportar los reportes a múltiples formatos

Los reportes generados con JasperReports pueden ser exportados a un gran número de formatos, incluyendo PDF (Formato de Documento Portable), XLS (Excel), RTF (Formato de Texto Enriquecido, el cual puede ser leído y editado por la mayoría de los procesadores de Word, incluyendo, pero no limitando a Microsoft Word, OpenOffice.org Writer, StarOffice Writer y WordPerfect), HTML (Lenguaje de Marcado de HiperTexto), XML (Lenguaje de Marcado Extensible), CSV (Valores Separados por Coma) y texto plano.

Existe además una tercera librería para exportar los reportes de JasperReports al formato OpenDocument Format (ODF). Este formato es una especificación de formato de archivo basado en un estándar XML para aplicaciones de oficina desarrollado por la organización para el avance de los estándares de estructuras de información (Organization for the Advancement of Structured Information Standards (OASIS)) OpenOffice.org versión 2.0 usa ODF como su formato por defecto.

JasperReports además brinda soporte para los Sistemas Operativos Windows y Linux.

Arquitectura de JasperReports

Como se muestra en la **figura 1.2**, la arquitectura de JasperReports está basada en archivos XML que por convención tienen la extensión *.jrxml*, los cuales contienen el diseño del reporte. El archivo diseñado será llenado por datos desde una base de datos, archivo XML, colecciones de java o modelos. Después de definir el diseño del reporte en el archivo *.jrxml* y determinar el origen de datos, JasperReports realiza el resto del trabajo. Compila el archivo de diseño y lo llena con los resultados del origen de datos para luego generar y exportar el reporte al formato elegido (PDF, Excel, HTML, XML, RTF, TXT, etc.) (16)

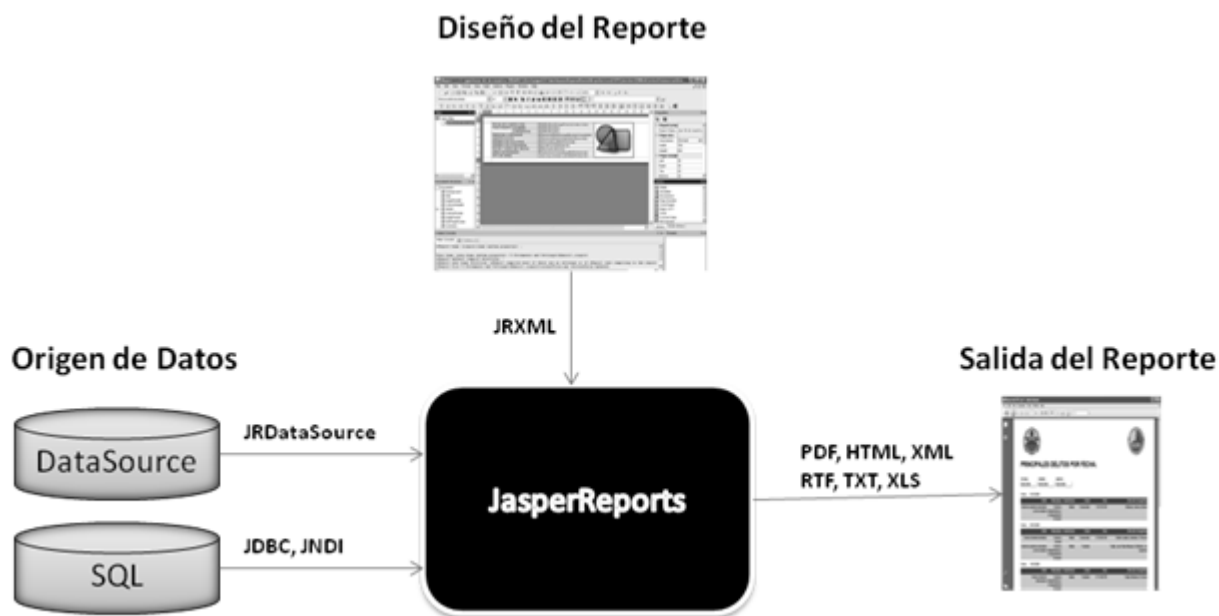


Figura 1.2: Arquitectura de JasperReports

Actualmente existen proyectos en la UCI que ya han utilizado esta herramienta y cuentan con la experiencia requerida para el desarrollo con la misma.

SELECCIÓN DE LA HERRAMIENTA

Teniendo en cuenta lo anterior, puede llegarse a la conclusión de que tanto Birt como JasperReports cuentan con las características idóneas para el desarrollo de una aplicación de gestión en java utilizando la arquitectura descrita. Sin embargo, es de vital importancia contar con personal experto en la Universidad que pueda servir de ayuda ante cualquier problema que pueda presentarse, es por esta razón que se seleccionó JasperReports como herramienta para la generación de reportes.

Ante esta selección se hizo necesario investigar sobre el uso de la librería jfreechart y la herramienta iReport, para la generación de gráficos y el diseño de reportes respectivamente.

JFreeChart

JFreeChart es una librería libre para la generación de gráficos 100% escrita en java, que facilita a los desarrolladores mostrar gráficos profesionales de alta calidad en sus aplicaciones. Entre sus funcionalidades incluye el soporte de un alto rango de tipos de gráficos, un diseño flexible que permite crear los gráficos tanto del lado del cliente como del servidor y soporte a varios formatos de salida. (17)

JFreeChart permite crear gráficos de pastel (figura 1.3), de barras (figura 1.4), de líneas (figura 1.5), de pila (figura 1.6), entre otros.

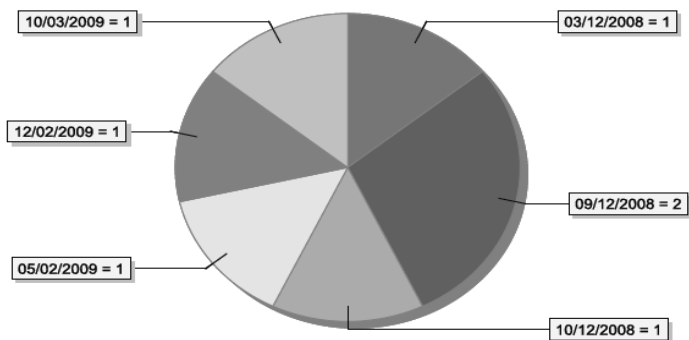


Figura 1.3: Gráfico de Pastel creado con la librería JFreeChart

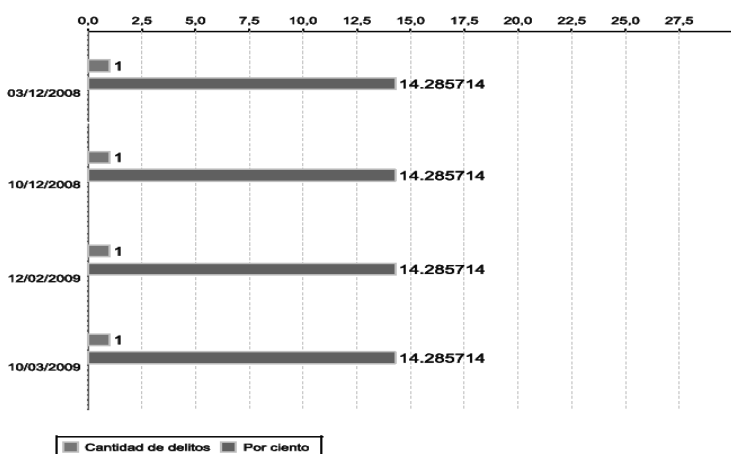


Figura 1.4: Gráfico de Barras creado con la librería JFreeChart

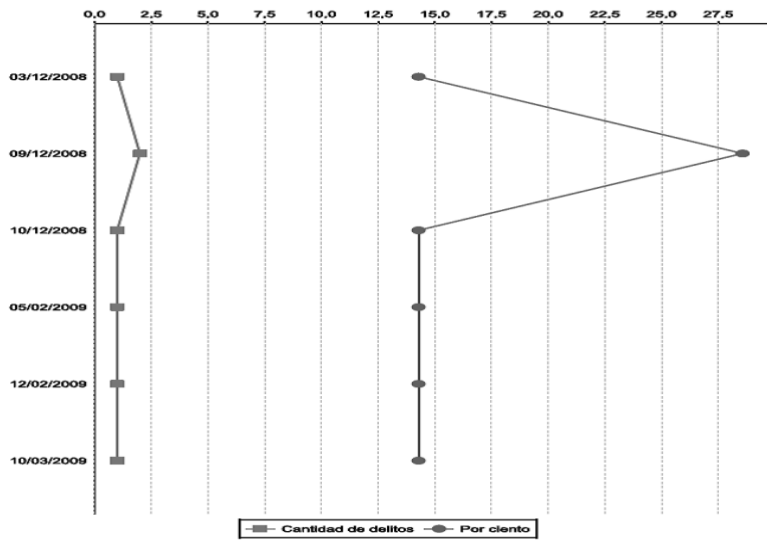


Figura 1.5: Gráfico de Líneas creado con la librería JFreeChart

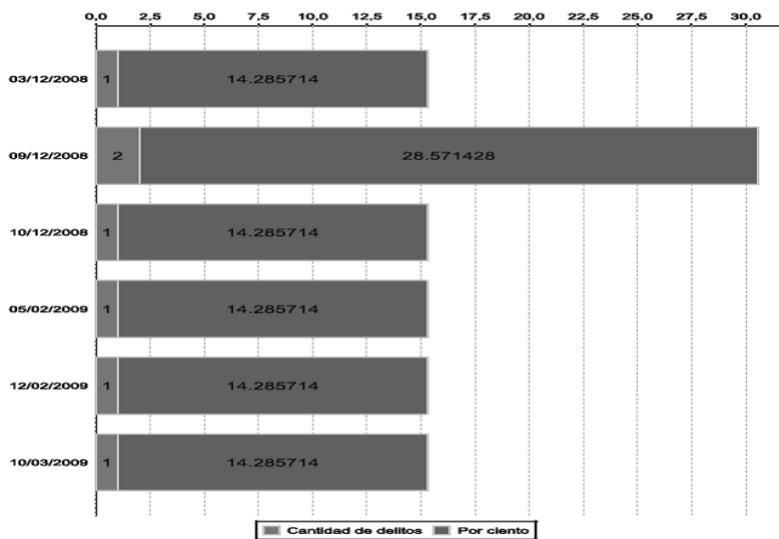


Figura 1.6: Gráfico de Pila creado con la librería JFreeChart

IREPORT

IReport es un constructor / diseñador de informes visual, poderoso, intuitivo y fácil de usar para JasperReports. Brinda todas las funciones más importantes para crear un excelente reporte en corto tiempo, ayuda a las personas que no conocen la librería JasperReports a crear reportes complejos y

aprender su sintaxis XML observando el código generado. Permite a los desarrolladores crear reportes realmente complejos, ahorrándoles gran cantidad de tiempo.

Entre las características más importantes de iReport se encuentran: (18)

- 100% escrito en JAVA y además open source y gratuito.
- Maneja el 98% de las etiquetas de JasperReports.
- Permite diseñar con sus propias herramientas: rectángulos, líneas, elipses, campos de los textfields y subreportes.
- Soporta internacionalización nativamente.
- Navegador de la estructura del documento.
- Recopilador y exportador integrados.
- Soporta JDBC.
- Soporta JavaBeans como orígenes de datos (éstos deben implementar la interface JRDataSource).
- Incluye Wizards (asistentes) para crear automáticamente informes.
- Tiene asistentes para generar los subreportes.
- Tiene asistentes para las plantillas.
- Facilidad de instalación.

Librerías

JasperReports necesita además de otras librerías para implementar algunas de sus funcionalidades, entre ellas:

- **commons-digester.jar**: La librería Common Digester incluye clases utilitarias que se usan para inicializar objetos de Java desde archivos XML. JasperReports aprovecha las ventajas del

componente Digester del repositorio Jakarta Commons para implementar sus funcionalidades de parsear XML. Esta librería debe encontrarse en el CLASSPATH de la aplicación para que JasperReports funcione correctamente.

- **commons-collections.jar**: Commons Collection es otro de los componentes de la suite de Jakarta Commons. Este componente brinda la funcionalidad de complementar el framework de colecciones de java. JasperReports aprovecha las ventajas de este componente para implementar algunas de sus funcionalidades y lo incluye en su directorio de librerías.
- **commons-logging.jar**: La librería Common Logging incluye componentes que apoyan al desarrollo con el envío de datos a archivos de autenticación. JasperReports aprovecha las ventajas de este componente y lo incluye en su directorio de librerías.
- **commons-beanutils.jar**: La última librería que requiere JasperReports para compilar los reportes es Common BeanUtils. BeanUtils es una librería que ofrece wrappers fáciles de usar a través de Java reflection e introspection API. JasperReports la incluye en su directorio de librerías.
- **itext-1.02b.jar**: Es la librería que se usa para generar y manipular PDF. Tiene además la habilidad de manipular documentos RTF, XML y HTML. JasperReports aprovecha las ventajas que ofrece iText para exportar los reportes a PDF y RTF.
- **poi-2.0-final-20040126.jar**: Jakarta POI es una librería de clases de java para crear y manipular varios formatos de Microsoft Office. JasperReports aprovecha las ventajas que ofrece poi para exportar los reportes a XLS (Excel). (15)

Estas herramientas y librerías, integradas a la librería de JasperReports, brindarán los medios necesarios para lograr una estructura que responda ante los requerimientos relacionados con reportes que deberá cumplir el proyecto CICPC.

1.3. METODOLOGÍAS DE DESARROLLO

Todo desarrollo de software es riesgoso y difícil de controlar, siéndolo aun más, cuando no se siga una guía que pauté los pasos a seguir y muestre cómo actuar ante las dificultades. Este papel lo juegan las metodologías de desarrollo de software. Sin ellas, el resultado de un proyecto sería desastroso, con clientes insatisfechos con el resultado, malas planificaciones con fechas límite imposibles de cumplir y desarrolladores aún más insatisfechos.

Sin embargo, aún aplicando una metodología determinada durante el proceso de desarrollo del software, no se logra obtener los resultados esperados. Una de las causas de este fenómeno viene dada por la variedad de metodologías que existe en la actualidad y la clasificación de las mismas. Son muchos los factores a tener en cuenta a la hora de seleccionar una metodología para que ésta se ajuste al ambiente de desarrollo y ayude, en vez de entorpecer, a obtener los resultados esperados.

Basándose en este planteamiento, se hizo un estudio de las mismas, para encontrar cuál se adaptaría mejor a una aplicación de pequeña escala con poco personal y requerimientos cambiantes. Para ello se analizaron RUP (Rational Unified Process), FDD (Feature Driven Development) y XP (Xtream Programming)

RUP

La metodología de desarrollo RUP forma parte de las llamadas metodologías pesadas. Entre sus principales características figuran que cuenta con una forma disciplinada de asignar tareas y responsabilidades, pretende implementar mejores prácticas, propone un desarrollo iterativo y el uso de la arquitectura basada en componentes además de un riguroso control de cambios, se basa en el modelado visual y le da gran importancia a la verificación de la calidad del software.

El proceso de ciclo de vida de RUP se divide en cuatro fases llamadas Inicio, Elaboración, Construcción y Transición. Estas fases se dividen en iteraciones, cada una de las cuales produce una pieza de software demostrable. (Figura 1.7)

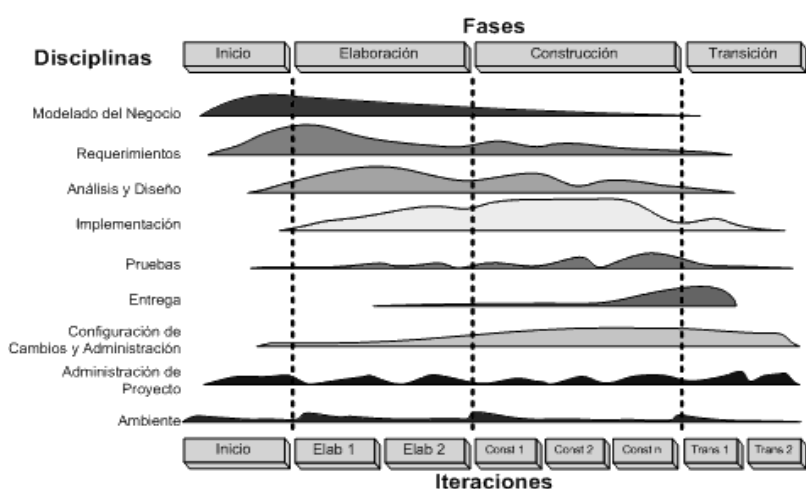


Figura 1.7: Fases y Flujos de trabajo en RUP

La metodología RUP es poco factible para ser utilizada en proyectos pequeños, pues puede que no sea posible cubrir los costos de dedicación del equipo de profesionales, ya que tendrían que invertir una gran cantidad de horas y esfuerzo para hacer las cosas según lo que dicen los procesos.

FDD

FDD con sus siglas en inglés Feature Driven Development es un enfoque ágil para el desarrollo de sistemas. Dicho enfoque no hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción. Sin embargo, fue diseñado para trabajar con otras actividades de desarrollo de software y no requiere la utilización de ningún modelo de proceso específico. Además, tiene en cuenta aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del avance del proyecto. Al contrario de otras metodologías, FDD afirma ser conveniente para el desarrollo de sistemas críticos.

Los principios de FDD son pocos y simples: (19)

- Se requiere un sistema para construir sistemas si se pretende escalar a proyectos grandes.
- Un proceso simple y bien definido trabaja mejor.
- Los pasos de un proceso deben ser lógicos y su mérito inmediatamente obvio para cada miembro del equipo.
- Vanagloriarse del proceso puede impedir el trabajo real.
- Los buenos procesos van hasta el fondo del asunto, de modo que los miembros del equipo se puedan concentrar en los resultados.
- Los ciclos cortos, iterativos y orientados por rasgos (features) son mejores.

Hay tres categorías de rol en FDD: roles claves, roles de soporte y roles adicionales. Los seis roles claves de un proyecto son: 1- administrador del proyecto, quien tiene la última palabra en materia de visión, cronograma y asignación del personal; 2- arquitecto jefe (puede dividirse en arquitecto de dominio y arquitecto técnico); 3- manager de desarrollo, que puede combinarse con arquitecto jefe o manager de proyecto; 4- programador jefe, que participa en el análisis del requerimiento y selecciona rasgos del conjunto a desarrollar en la siguiente iteración; 5- propietarios de clases, que trabajan bajo

la guía del programador jefe en diseño, codificación, prueba y documentación, repartidos por rasgos y 6- experto de dominio, que puede ser un cliente, patrocinador, analista de negocios o una mezcla de todo eso.

FDD consiste en cinco procesos secuenciales durante los cuales se diseña y construye el sistema. La parte iterativa soporta desarrollo ágil con rápidas adaptaciones a cambios en requerimientos y necesidades del negocio. Cada fase del proceso tiene un criterio de entrada, tareas, pruebas y un criterio de salida. (Figura 1.8)

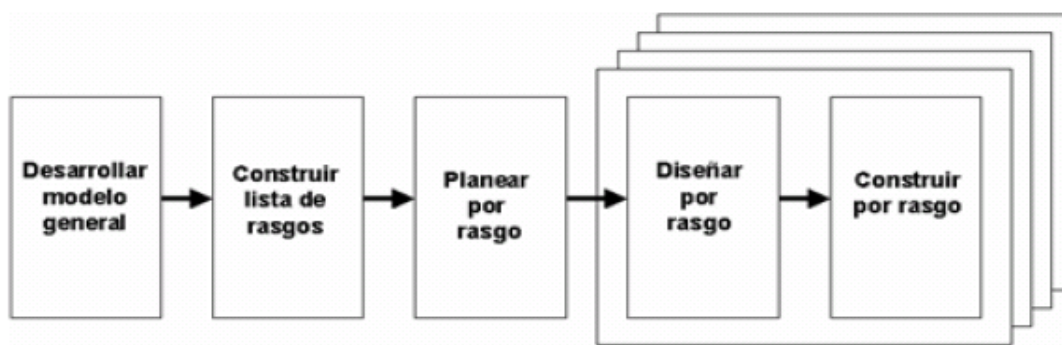


Figura 1.8: Proceso FDD

XP

La programación extrema se basa en la simplicidad, la comunicación y el reciclado continuo de código. Promueve los valores de comunicación directa entre las personas; el coraje de exponer sus dudas, miedos, experiencias sin "embellecer" éstas de ninguna de las maneras; el intento de mantener el software lo más simple posible y la capacidad de respuesta ante los cambios que se van haciendo necesarios a lo largo del camino.

Los objetivos de XP son muy simples: la satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, se debe responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final del ciclo de la programación. (20)

El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software.

El ciclo de vida de XP se enfatiza en el carácter iterativo e incremental del desarrollo. Sus iteraciones son relativamente cortas ya que se piensa que entre más rápido se le entreguen resultados de valor al cliente, más retroalimentación se va a obtener, lo cual representará una mejor calidad del producto a largo plazo. El mismo se encuentra representado en la imagen 1.10.

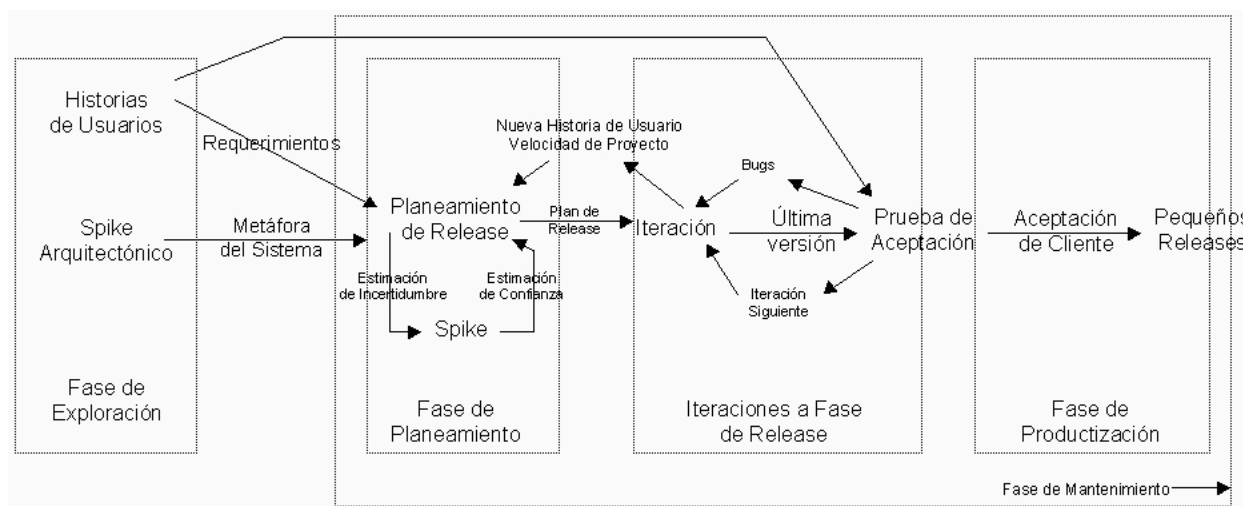


Figura 1.9: Ciclo de Vida de XP

SELECCIÓN DE LA METODOLOGÍA

Para la selección de la metodología de desarrollo a utilizar, se hace necesario analizar el entorno en el que se construirá el producto.

En primer lugar es necesario destacar que los usuarios finales son los propios desarrolladores del proyecto CICPC, pues son los que tienen la necesidad de un componente que los abstraiga de toda la lógica relacionada con reportes, lo cual significa que en todo momento se estará interactuando con los mismos.

Otro de los factores es que los requerimientos asociados a los reportes estarán cambiando constantemente, pues aunque se parte de requerimientos ya especificados, el propio proceso de desarrollo del proyecto CICPC arrojará nuevas necesidades y por ende nuevas funcionalidades a incluir en el componente.

Además, se hace necesario una solución rápida, pues el componente que se propone formará parte de los cimientos del sistema de CICPC y por lo tanto deberá estar funcional desde el momento en que dicho proyecto inicie su proceso de desarrollo.

Aparejado a lo antes descrito se encuentra el factor de que se cuenta con poco personal de desarrollo, lo cual dificulta la adopción de roles en cada etapa así como la generación de múltiples artefactos.

Una vez que el componente sea añadido a la arquitectura de CICPC, el mismo será sometido a constantes pruebas por parte de los desarrolladores, lo cual significa que se le estará dando mantenimiento al código a la vez que se desarrolle el resto de las iteraciones identificadas.

Lo antes descrito evidencia cómo las características del presente trabajo se inclinan a adoptar las prácticas y valores de la metodología XP por lo cual se seleccionó la misma como rectora del proceso de desarrollo del software

Los pasos planteados por XP fueron moldeados a las características especiales del desarrollo. Se contemplaron cinco fases en el proceso: exploración, planeación, iteraciones a fase de release, productización y mantenimiento. La fase restante que se contempla en el ciclo de vida de XP es muerte, la cual no es aplicable pues se espera que aún después de cumplidos los objetivos del presente trabajo surjan nuevas funcionalidades a implementar.

Se adoptaron los valores de XP comunicación, simplicidad, retroalimentación, coraje y respeto. La comunicación entre el equipo de desarrollo y los clientes será maximizada y efectuada de forma directa e interpersonal con el fin de evitar los problemas y errores causados por la mala adopción de la misma. Se realizará un trabajo tan simple como sea posible, favoreciendo la comunicación, reduciendo el código sin utilizar y garantizando la calidad. Se estará constantemente midiendo el sistema para conocer cuánto se acerca a las funcionalidades necesarias mediante pruebas automáticas y por parte de los usuarios. Los valores antes mencionados contribuirán a enfrentar con coraje cada cambio en los requerimientos y refactorizaciones al sistema. Todo esto implica respeto hacia la organización y las personas cuya vida cambiará con el componente que se está proponiendo.

Entre las reglas y prácticas propuestas por XP se tendrán en cuenta las siguientes:

Serán escritas historias de usuario, se creará un plan de entregables basado en realizar frecuentemente pequeñas entregas y el proyecto será dividido en iteraciones las cuales serán planeadas en cada comienzo. Se realizará un diseño simple utilizando tarjetas CRC en las sesiones a

realizar, se crearán soluciones spike para reducir el riesgo y se refactorizará siempre que sea posible. El cliente siempre estará disponible, se escribirán pruebas de unidad, se integrará continuamente y no se trabajará fuera de tiempo. El código estará respaldado por pruebas de unidad y no podrá ser entregado mientras las mismas no tengan éxito y una vez que el producto sea entregado se realizarán constantemente pruebas de aceptación.

Con la adopción de las características antes mencionadas se espera llevar a cabo un proceso de desarrollo que guíe los pasos hacia la construcción de un producto de calidad, coste aceptable y que cumpla con los plazos de tiempo y alcance previstos.

1.4. CONCLUSIONES

En el presente capítulo se analizaron los conceptos principales asociados a la investigación para seleccionar las herramientas de trabajo, las librerías de clases y las metodologías de desarrollo adecuadas para la construcción del producto final.

Después de estudiada la bibliografía sobre las herramientas de generación de reportes existentes, se decidió utilizar la librería JasperReports por las facilidades que brinda, así como todas aquellas de las que depende para su correcto funcionamiento. Además se estudiaron las metodologías pertenecientes a los diferentes enfoques de desarrollo de software, para arribar a la conclusión de que el proceso que más se adapta al presente trabajo es el de XP, eligiéndose como metodología rectora del proceso de desarrollo.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

Una vez realizado el análisis del estado del arte, elegidas las herramientas y la metodología de desarrollo a utilizar; se está en condiciones de plantear una propuesta de solución al problema descrito.

La solución que se propone constituye un componente arquitectónico que se pueda insertar como una pieza de rompecabezas dentro de la arquitectura a utilizar en el proyecto CICPC. Dicho componente incluirá las funcionalidades comunes a la generación de reportes dentro del proyecto, además de otras de apoyo a las anteriores. Se espera como resultado un componente flexible a cambios que puedan surgir y que facilite a gran escala el trabajo a aquellos que lo utilicen.

La propuesta queda resumida en la imagen 2.1. Dicho componente estará compuesto por un conjunto de clases que interactuarán para generar los reportes. Las clases utilizarán para su funcionamiento la librería JasperReports. En la imagen se representa el resto de las librerías que garantizan el correcto desarrollo del componente. Los diseños de los reportes estarán contenidos en el directorio WebContent/resources cuya estructura interna estará formada por paquetes correspondientes a cada uno de los módulos del proyecto y estos a su vez contendrán los submódulos que lo forman. Serán obtenidos haciendo uso del contexto de JSF, por lo cual el componente que se propone estará fuertemente ligado a este framework.

El presente capítulo se encarga de definir las funcionalidades necesarias a incluir en dicha propuesta de solución así como la estrategia a seguir para la implementación de la misma adecuándose a los plazos de tiempo impuestos por los usuarios. Además resume los resultados del estudio de las herramientas para la generación de reportes a utilizar y provee de los artefactos obtenidos a raíz de la aplicación de la metodología de desarrollo seleccionada.

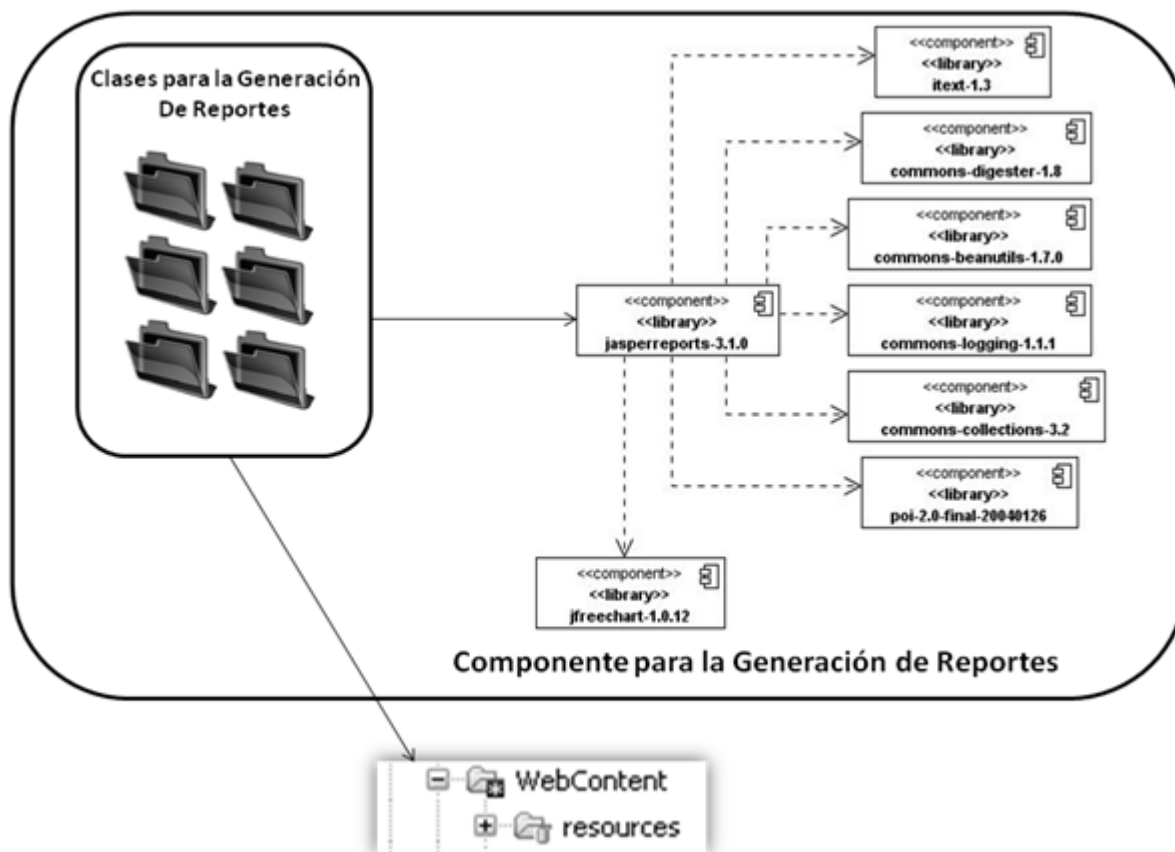


Figura 2.1: Propuesta de Solución

2.1. EXPLORACIÓN

Para comenzar la producción de un producto de software es necesario asegurarse de que es factible y posible hacerlo. Se debe tener confianza en que las herramientas seleccionadas ayudarán a la culminación del trabajo. Se debe creer que una vez que el código se haga, éste puede utilizarse cada día. Cada miembro del equipo debe confiar en sus propias habilidades y en las habilidades de los otros.

La fase de exploración ayuda a resolver todos estos conflictos. Es en ella donde los clientes plantean a grandes rasgos las historias de usuario que son de interés para el producto. Durante la misma, los desarrolladores interactúan con las herramientas a utilizar durante todo el proceso, exploran activamente las posibilidades de arquitectura y experimentan los límites de las tecnologías a utilizar.

(21)

Durante la fase de exploración el equipo de desarrollo procedió a realizar un estudio en profundidad en cuanto a las herramientas y librerías de desarrollo seleccionadas para explorar las posibles soluciones a generar.

CICLO DE VIDA DE JASPERREPORTS

Para un mejor entendimiento de las librerías de desarrollo seleccionadas, se estudiaron los estados por los que pasa un reporte en el proceso de generación. **Figura 2.2**

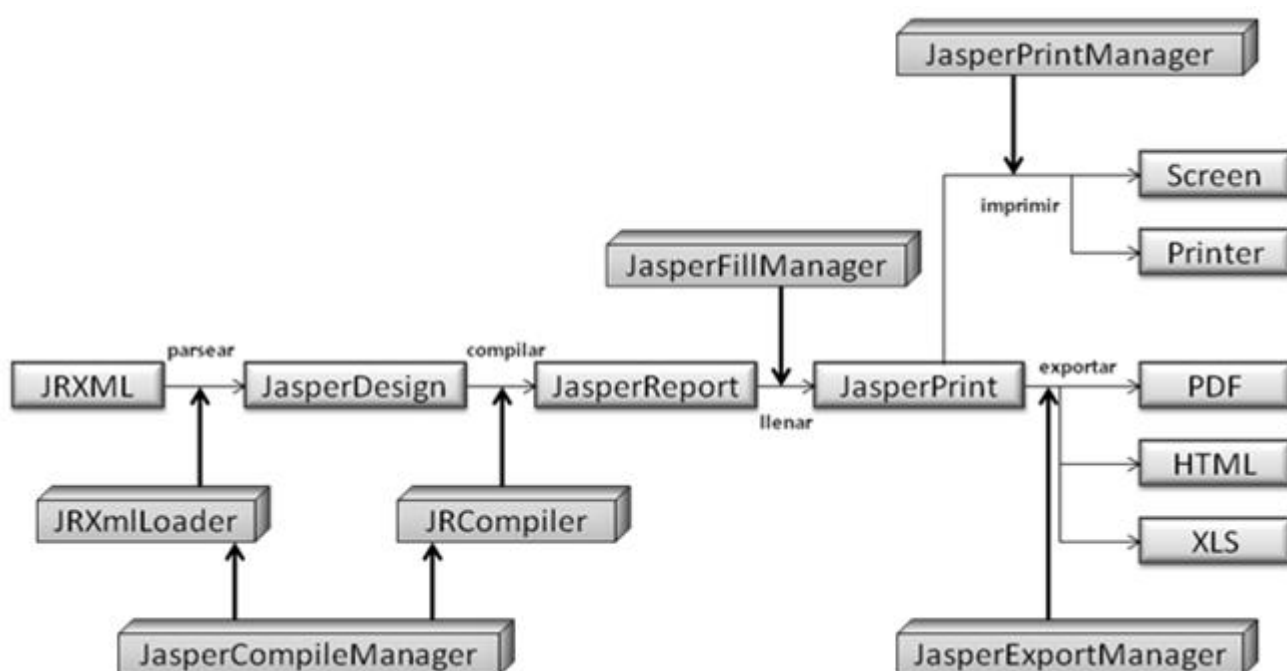


Figura 2.2: Ciclo de Vida del Reporte

Como se muestra en la figura y se explicó en el capítulo anterior, el proceso de generación de reportes parte de un archivo *.jrxml*, el cual pertenece a la familia de XML basado en el DTD de JasperReports y que contiene el diseño del reporte a generar.

El DTD de JasperReports es el siguiente:

```
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
```

Un ejemplo de archivo *.jrxml* se muestra en la **figura 2.3**:

```

- <!--
  Created with iReport - A designer for JasperReports
-->
- <jasperReport name="simple_template" columnCount="1" printOrder="Vertical" orientation="Portrait" pageWidth="595" pageHeight="842"
columnWidth="535" columnSpacing="0" leftMargin="30" rightMargin="30" topMargin="20" bottomMargin="20" whenNoDataType="NoPages"
isTitleNewPage="false" isSummaryNewPage="false">
  <property name="ireport.scriptlethandling" value="2"/>
  <property name="ireport.encoding" value="UTF-8"/>
  <import value="java.util.*/>
  <import value="net.sf.jasperreports.engine.*/>
  <import value="net.sf.jasperreports.engine.data.*/>
- <background>
  <band height="0" isSplitAllowed="true"> </band>
</background>
- <title>
  <band height="50" isSplitAllowed="true"> </band>
</title>
- <pageHeader>
  <band height="50" isSplitAllowed="true"> </band>
</pageHeader>
- <columnHeader>
  <band height="30" isSplitAllowed="true"> </band>
</columnHeader>
- <detail>
  <band height="100" isSplitAllowed="true"> </band>
</detail>
- <columnFooter>
  <band height="30" isSplitAllowed="true"> </band>
</columnFooter>
- <pageFooter>
  <band height="50" isSplitAllowed="true"> </band>
</pageFooter>
- <lastPageFooter>
  <band height="50" isSplitAllowed="true"> </band>
</lastPageFooter>
- <summary>
  <band height="50" isSplitAllowed="true"> </band>
</summary>
</jasperReport>

```

Figura 2.3: Archivo *.jrxml*

Este ejemplo ilustra los principales elementos de un archivo *.jrxml*, todos son opcionales con la excepción del elemento raíz `<jasperReport>`. Cada uno de estos elementos contiene como hijo un elemento `<band>`. Las bandas contienen los datos que se muestran en el reporte.

La función de cada uno de los elementos presentados es la siguiente:

Title. Esta es la primera sección del reporte, generada solo una vez durante el proceso de llenado del reporte y ubicada al comienzo del documento resultante.

PageHeader. Aparece en la parte superior de cada página del documento resultante.

ColumnHeader. Aparece al inicio de cada columna definida en el reporte

Detail. En esta sección es donde se encuentran los datos principales que contendrá el reporte. Para cada tupla del origen de datos, esta sección será generada.

ColumnFooter. Aparece al terminar cada columna.

PageFooter. Aparece en la parte inferior de cada página.

Summary. Esta sección es generada sólo una vez por reporte y aparece al final del documento generado, pero no es necesariamente la última sección, porque en algunos casos el columnFooter y el pageFooter de la última página lo sobrevienen.

Haciendo uso de la clase JRXMLLoader el archivo *.jxml* es parseado y convertido en un objeto JasperDesign. Dicho objeto mantiene la estructura del reporte descrita anteriormente, pero en forma de objetos. Brinda la posibilidad de insertar, modificar o eliminar cualquier elemento contenido en el reporte.

El objeto JasperDesign representa la estructura del reporte a través de dos clases, JRBaseReport, que mediante una jerarquía de clases contiene todos los elementos del reporte, dígase las bandas mencionadas anteriormente y éstas a su vez los distintos elementos que pueden contener como textos estáticos, textos dinámicos, gráficos, tablas, entre otros. JasperDesign hereda directamente de JRBaseReport y por lo tanto contiene todos los elementos del mismo. La otra clase sería JRDesignDataSet que contiene las expresiones, parámetros, campos, variables y grupos que componen el reporte. JasperDesign contiene un elemento de tipo JRDesignDataSet. La relación descrita se puede apreciar en la **figura 2.4**.

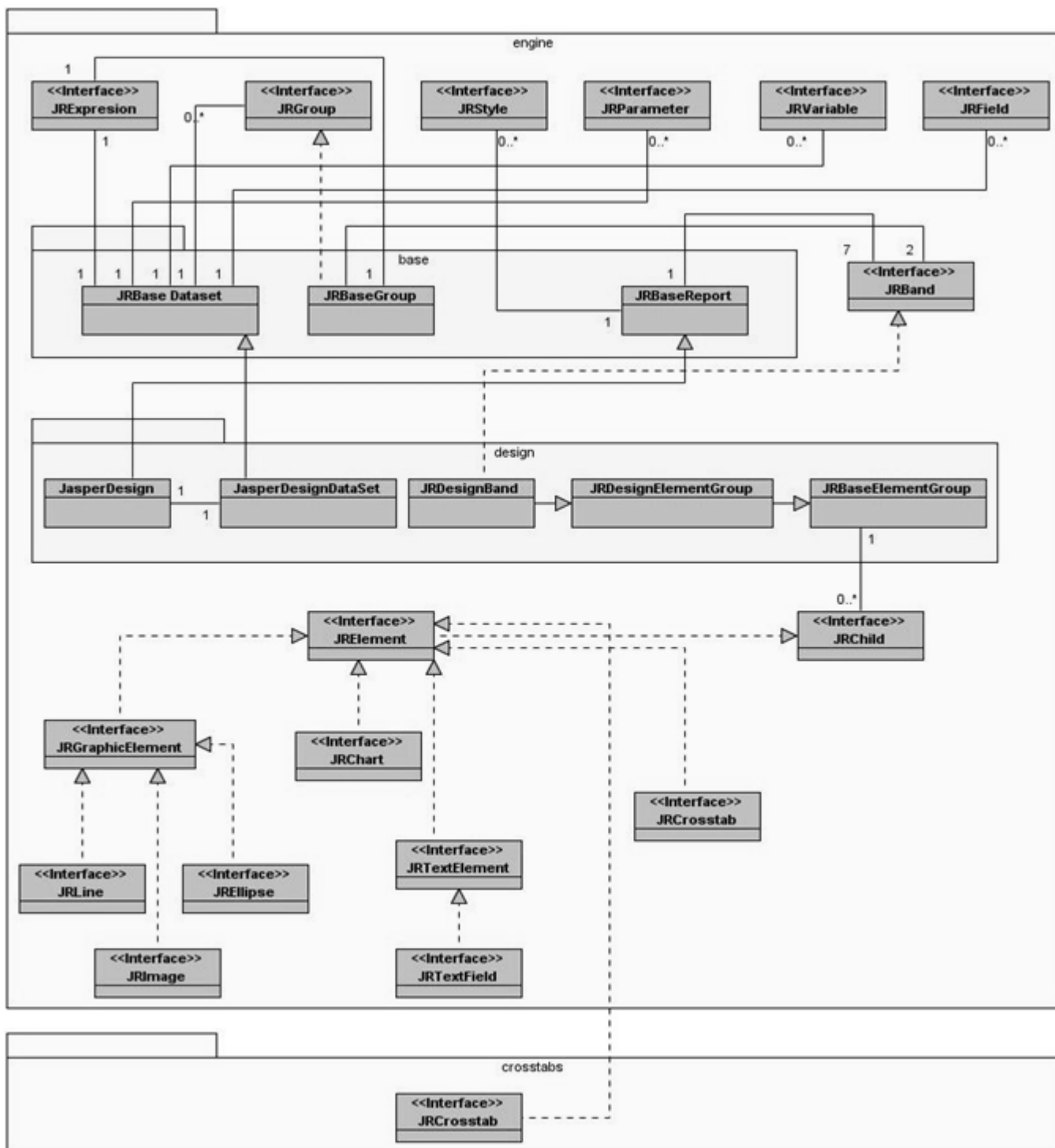


Figura 2.4: Diseño de clases que componen el objeto JasperDesign

Cada una de las clases representadas en el diagrama forma parte, como se observa, del objeto de tipo JasperDesign. Existen otras clases y relaciones que también forman parte del mismo; pero que desde el punto de vista funcional no intervienen en el problema que nos planteamos.

Haciendo uso de la clase JRCompiler, el objeto de tipo JasperDesign es compilado y convertido en un objeto de tipo JasperReport. Es en este momento que se tiene conocimiento de los posibles errores de sintaxis que pueda tener el reporte, pues si existe alguno este proceso lanza una excepción de tipo JRException. Es válido aclarar que el objeto JasperReport no se obtiene solamente siguiendo estos pasos, otra vía sería haciendo uso de la clase JasperCompileManager, donde se obtendría directamente el objeto a partir del archivo *.jrxml*, sin embargo, esta vía no es factible por las muchas funcionalidades que se pueden adicionar haciendo uso de las prestaciones del objeto intermedio JasperDesign.

La otra opción consiste en compilar directamente el reporte en la herramienta para el diseño iReport, esta compilación daría como resultado un archivo de extensión *jasper* que sería el sustituto en este caso del archivo *.jrxml* y a partir del cual se continuaría el proceso después de la compilación; esta vía tampoco es factible, pues en el diseño del reporte es necesario especificar el tipo de los campos que se van a mostrar, ya sea String, Integer, etc.; teniendo esto en cuenta, existe algunos campos que se deben mostrar en los reportes que responden a tipos de clases específicos de la aplicación que se va a desarrollar, y por tanto la librería de JasperReports no reconoce, generando así errores en el proceso de compilación.

Ya teniendo el reporte compilado en el objeto JasperReport y haciendo uso de la clase JasperFillManager, se procede al llenado del reporte a un objeto de tipo JasperPrint. Para ello, es necesario crear una clase intermedia que implemente la interfaz JRDataSource. Éstas son clases especiales usadas para pasar los datos al reporte.

Una vez llenado el reporte ya se puede imprimir por pantalla o en una impresora haciendo uso de la clase JasperPrintManager, estas funcionalidades no serán tenidas en cuenta debido a que no constituyen requisitos del software.

El objeto JasperPrint además de imprimirse se puede exportar a diferentes formatos haciendo uso de la clase JasperExportManager, lo cual constituye el momento final en el ciclo de vida de los reportes.

HISTORIAS DE USUARIO

Las historias de usuario son la unidad de funcionalidad en un proyecto XP. Se demuestra el progreso aplicando pruebas e integrando código que implementa la historia. La historia de usuario debe ser fácil de entender tanto por desarrolladores como clientes; posible de probar, de valor para el cliente y lo suficientemente pequeña para que los programadores puedan construir un considerable grupo de ellas en cada iteración. (21)

Durante esta fase los clientes realizaron estudios del levantamiento de requisitos del proyecto CICPC para derivar de los mismos las historias de usuario, se obtuvieron los siguientes resultados:

1. Generar Reporte

Al seleccionar la opción para generar reporte, se debe procesar la lista de datos haciendo uso de un origen de datos generalizado que posibilite mostrar textos con estilo en el reporte, generar el reporte atendiendo a un formato previo definido y mezclado con el diseño original y exportar el mismo a los formatos PDF o XLS en dependencia de la selección del usuario.

2. Generar gráficos.

Al seleccionar la opción para generar un gráfico, se debe interpretar la información enviada por el usuario y generar un gráfico que puede ser de los siguientes tipos: pastel, pila, barras y área, en dependencia de la selección del usuario, para luego insertarlo como un elemento de una banda específica del reporte.

3. Modificar formato de reportes estadísticos.

Al seleccionar la opción para modificar el formato de un reporte deberá cambiar en el reporte los atributos de color, texto y fondo tanto de los elementos de texto como de las celdas de las tablas contenidos en el reporte según la selección realizada por el usuario. Además de agrupar dinámicamente los datos en el reporte, eliminar secciones del mismo que contengan información no requerida por el usuario y modificar atributos de posicionamiento a elementos del reporte para que se ajusten a su formato.

4. Concatenar reportes.

Al seleccionar la opción concatenar reportes se debe generar un reporte a partir de un grupo de reportes dado.

2.2. PLANEACIÓN

Planeamos para garantizar que siempre estemos haciendo lo más importante que queda por hacer, a fin de mantenernos coordinados de manera efectiva con otras personas y para responder rápidamente a eventos inesperados. (22)

El propósito de la fase de planeación es establecer un acuerdo entre los clientes y desarrolladores sobre el menor tiempo en que la mayor cantidad de historias de usuario puedan ser realizadas. Para ello se realizó el juego de planeación con el objetivo de maximizar el valor del software producido a partir de la puesta en producción de las características más importantes lo antes posible, en el mismo participan en conjunto desarrolladores y clientes con el fin de determinar rápidamente el alcance de la versión a construir; esto se logra además gracias a tener al cliente siempre disponible, lo cual facilita la interacción continua entre los involucrados. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

A partir de las especificaciones anteriores se procedió a seguir el proceso representado por la **figura 2.5**:

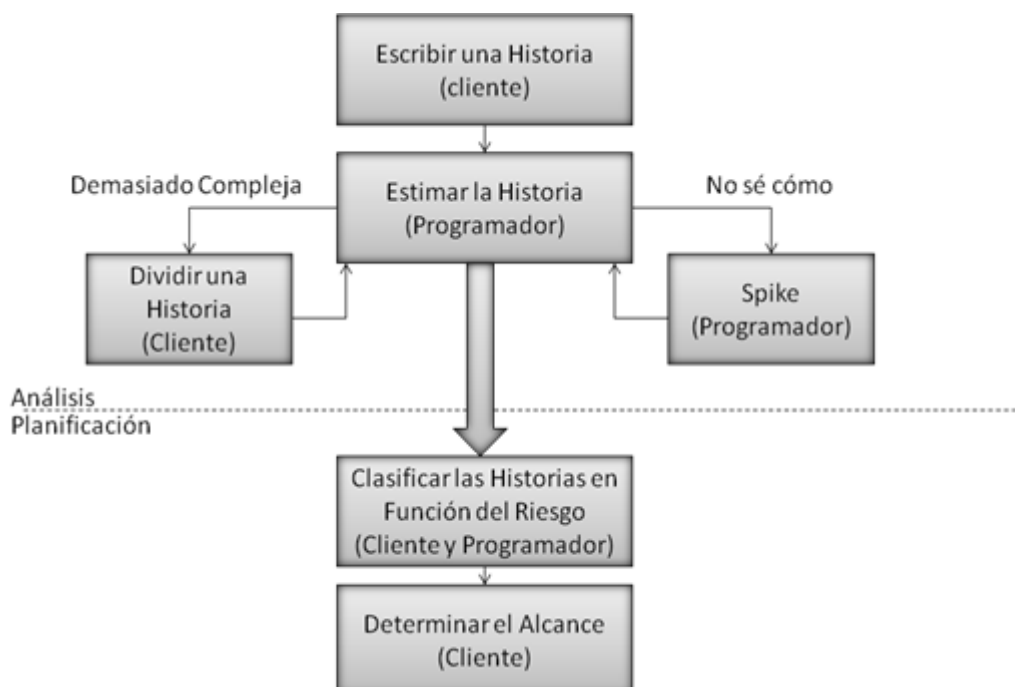


Figura 2.5: Proceso seguido en la fase de planeación.

Ya el cliente escribió las historias de usuario, por lo tanto, es tarea del programador realizar la estimación de las mismas. Teniendo en cuenta que por primera vez se enfrenta a una tecnología de este tipo, se hizo necesario por cada una de las historias anteriores realizar un spike para tener una noción a grandes rasgos de la complejidad de las tareas asociadas a la historia.

El uso del spike dio como resultado que las historias 1 y 3 sería demasiado complejo realizarlas como un todo, se le explicó al cliente que en ambos casos se engloban funcionalidades que por sí solas pueden necesitar un gran esfuerzo de implementación y la necesidad de dividirlos. El cliente procedió entonces a realizar la división.

Luego de la división, ya se estuvo en condiciones de realizar una estimación de las nuevas historias de usuario, lo cual se llevó a cabo a partir del conocimiento existente sobre las tecnologías a utilizar y las características de las mismas. Es válido aclarar que dicha estimación es necesario irla adaptando a través del proceso de desarrollo, puesto que para su realización no se tuvo un punto de partida; sin embargo se espera que la repetición de dicho proceso en el futuro arroje resultados más reales a partir de la experiencia adquirida.

Luego se procedió a ordenar las historias según el momento de implementación, para lo cual se tuvo en cuenta su valor para el negocio, el riesgo técnico que representan y las dependencias con otras historias.

Las historias más importantes son aquellas que contienen un mayor valor para el negocio. Se debe tener cuidado con la secuencia de historias basadas en dependencias técnicas, pues la mayoría de las veces estas dependencias son menos importantes que el valor. (22)

Las historias de usuario de mayor valor para el negocio, según las necesidades del cliente, son aquellas que incluyen funcionalidades imprescindibles para seguir todo el proceso de generación de reportes, aún cuando los mismos no cuenten con elementos finales como gráficos y formatos.

Al analizar el riesgo técnico de las historias de usuario se arribó a la conclusión de que todas presentaban riesgo técnico, pues requerían un profundo conocimiento por parte del desarrollador de la lógica asociada a los reportes así como de las herramientas técnicas a utilizar.

Sobre las dependencias entre ellas, se concluyó que se partiría de la historia “generar reporte”, para a través de la misma construir todas las demás funcionalidades.

Posteriormente se procedió a realizar un plan de entregas entre desarrolladores y clientes basado en los datos obtenidos en el proceso anterior. Una vez que el desarrollador estimó la historia de usuario y proporcionó al cliente un tiempo estimado de construcción de la misma; el cliente estuvo en condiciones de decidir cuánto tiempo debe durar cada iteración a realizar y cuáles historias se implementarán en cada una de ellas. Los puntos estimados se refieren a días, teniendo en cuenta que la mayoría de las historias tienen una duración menor a una semana.

Una vez obtenidos todos estos datos, las historias de usuario quedaron de la siguiente forma:

Historia de Usuario	
Número: 1	Usuario: Desarrollador de CICPC
Nombre historia: Generar Reporte	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 5	Iteración asignada: 1
Descripción: Al seleccionar la opción para generar reportes, se debe procesar la lista de datos y generar el reporte.	
Observaciones: La lista de datos debe ser procesada como un origen de datos de valor para JasperReports	

Historia de Usuario	
Número: 2	Usuario: Desarrollador de CICPC
Nombre historia: Embeber reporte en un formato determinado	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 5	Iteración asignada: 1
Descripción: Dar al reporte a generar un formato definido con anterioridad. Los formatos definidos para los reportes del CICPC varían con respecto al fin con el que sean generados. Los que se identificaron son los siguientes:	
<ul style="list-style-type: none"> ➤ Consultas: Incluye a todos los reportes que muestran listados de consultas realizadas en el sistema. ➤ Formularios: Incluye a todos los reportes que muestran características de una entidad dada. 	

- **Memorándum:** Incluye a todos los reportes que representan informes que circulan dentro de la organización.
- **Oficio:** Incluye a todos los reportes que representan informes que serán enviados fuera de la organización.
- **Estadísticas:** Incluye a todos los reportes que muestran datos estadísticos de la organización.
- **Informe de Criminalística:** Incluye a todos los reportes que representan informes pertenecientes a la Coordinación Nacional de Criminalística.
- **Informe de Forense:** Incluye a todos los reportes que representan informes pertenecientes a la Coordinación Nacional en Ciencias Forenses.
- **Informe de Investigación Interna:** Incluye a todos los reportes que representan informes pertenecientes al departamento de Investigación Interna. Algunas de estas plantillas se encuentran definidas en el [anexo 1](#).

Observaciones: Para embeber el reporte en su formato correspondiente se debe crear el mismo como archivo `.jspxml` y mezclarlo una vez obtenido el objeto JasperDesign.

Historia de Usuario	
Número: 3	Usuario: Desarrollador de CICPC
Nombre historia: Parsear Cadena	
Prioridad en negocio: Media	Riesgo en desarrollo: Alto
Puntos estimados: 5	Iteración asignada: 2
Descripción: Tomar un texto en formato HTML y convertirlo en un formato que entienda el compilador de reportes.	
Observaciones: El PDF debe ser tratado individualmente por contener un tipo diferente de codificación.	

Historia de Usuario	
Número: 4	Usuario: Desarrollador de CICPC
Nombre historia: Exportar reporte a formato PDF	

Prioridad en negocio: Alta	Riesgo en desarrollo: Medio
Puntos estimados: 2	Iteración asignada: 1
Descripción: Tomar el reporte generado y exportarlo a formato PDF	
Observaciones:	

Historia de Usuario	
Número: 5	Usuario: Desarrollador de CICPC
Nombre historia: Exportar reporte a formato XLS	
Prioridad en negocio: Alta	Riesgo en desarrollo: Medio
Puntos estimados: 2	Iteración asignada: 1
Descripción: Tomar el reporte generado y exportarlo a formato XLS	
Observaciones:	

Historia de Usuario	
Número: 6	Usuario: Desarrollador de CICPC
Nombre historia: Generar Gráfico	
Prioridad en negocio: Baja	Riesgo en desarrollo: Alto
Puntos estimados: 5	Iteración asignada: 2
<p>Descripción: Al seleccionar la opción para generar un gráfico, se debe interpretar la información enviada por el usuario y generar un gráfico en dependencia de la selección del usuario.</p> <p>Los gráficos que se deben generar son los siguientes:</p> <ul style="list-style-type: none"> ➤ Gráfico de Pastel ➤ Gráfico de Pastel 3D ➤ Gráfico de Pila ➤ Gráfico de Barras ➤ Gráfico de Barras 3D ➤ Gráfico de Líneas ➤ Gráfico de Área 	

Observaciones:

Historia de Usuario	
Número: 7	Usuario: Desarrollador de CICPC
Nombre historia: Cargar Imagen	
Prioridad en negocio: Alta	Riesgo en desarrollo: Bajo
Puntos estimados: 1	Iteración asignada: 1
Descripción: Cargar las imágenes necesarias para ser incluidas en el reporte.	
Observaciones: Las imágenes deben encontrarse en el mismo directorio que los diseños del reporte.	

Historia de Usuario	
Número: 8	Usuario: Desarrollador de CICPC
Nombre historia: Concatenar Reporte	
Prioridad en negocio: Media	Riesgo en desarrollo: Bajo
Puntos estimados: 1	Iteración asignada: 2
Descripción: Al seleccionar la opción concatenar reportes se debe generar un reporte a partir de un grupo de reportes dado.	
Observaciones:	

Historia de Usuario	
Número: 9	Usuario: Desarrollador de CICPC
Nombre historia: Dar formato a elemento	
Prioridad en negocio: Baja	Riesgo en desarrollo: Bajo
Puntos estimados: 1	Iteración asignada: 2
Descripción: Al seleccionar la opción para dar estilo a los elementos de texto de un reporte se deberá cambiar en el reporte los atributos de color, texto y fondo de los elementos según la selección realizada por el usuario.	

Dichos atributos variarán en función de juegos de colores, el usuario no puede seleccionar lo que desee, sino un juego de colores en consecuencia con los representativos de la organización.

Económico



Clásico



Lujo



Observaciones:

Historia de Usuario	
Número: 10	Usuario: Desarrollador de CICPC
Nombre historia: Agrupar datos dentro del reporte	
Prioridad en negocio: Baja	Riesgo en desarrollo: Alto
Puntos estimados: 4	Iteración asignada: 2
Descripción: Al seleccionar la opción de agrupar los datos en el reporte se deberá agrupar los mismos según el criterio seleccionado por el usuario.	
Observaciones:	

Historia de Usuario	
Número: 11	Usuario: Desarrollador de CICPC
Nombre historia: Dar formato a tabla	
Prioridad en negocio: Baja	Riesgo en desarrollo: Bajo
Puntos estimados: 2	Iteración asignada: 2
Descripción: Al seleccionar la opción para dar estilo a las tablas de un reporte se deberá cambiar en el reporte los atributos de color, texto y fondo de las celdas de las	

<p>tablas según la selección realizada por el usuario.</p> <p>Dichos atributos variarán en función de juegos de colores, el usuario no puede seleccionar lo que desee, sino el que desee tener en consecuencia con los representativos de la organización, los cuales se corresponden con los de la historia de usuario 9.</p>
Observaciones:

Historia de Usuario	
Número: 12	Usuario: Desarrollador de CICPC
Nombre historia: Reposicionar datos	
Prioridad en negocio: Baja	Riesgo en desarrollo: Bajo
Puntos estimados: 2	Iteración asignada: 2
<p>Descripción: Al seleccionar la opción para reposicionar los datos dentro de un reporte se deberá eliminar secciones del mismo que contengan información no requerida por el usuario y modificar atributos de posicionamiento a elementos del reporte para que se ajusten a su formato.</p>	
Observaciones:	

Se identificaron dos iteraciones. La primera iteración, con duración de 3 semanas, se propone construir la infraestructura necesaria para generar los reportes de todo el sistema de CICPC, que incluye las funcionalidades básicas para el paso del reporte por todo su ciclo de vida y su presentación al usuario en el formato requerido (figura 2.6).

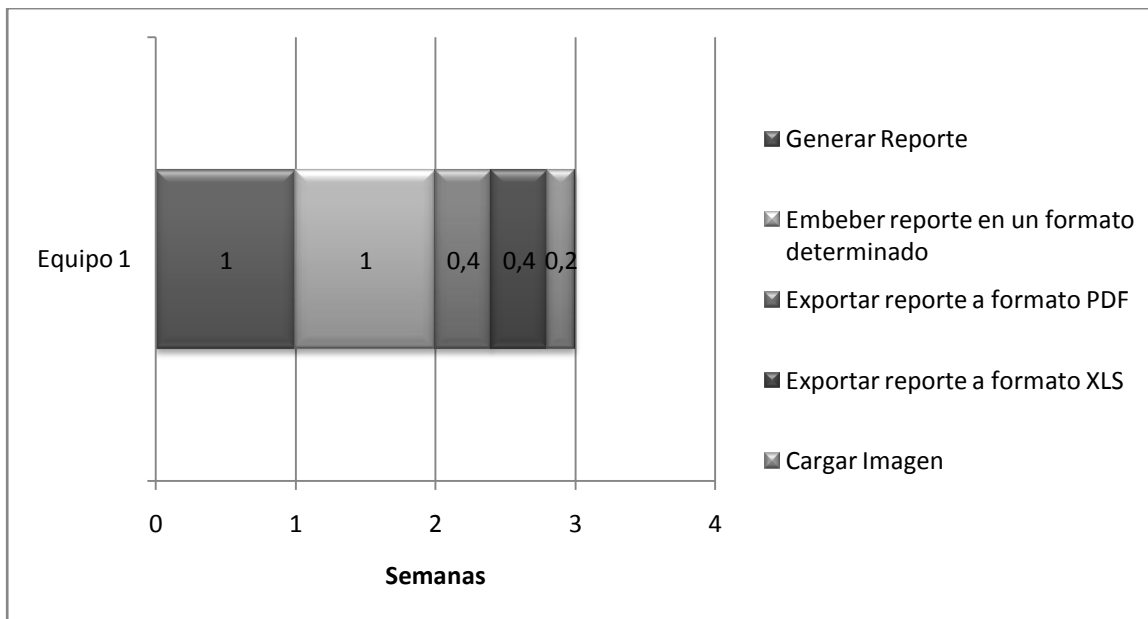


Figura 2.6: Primera Iteración.

La segunda iteración, con duración de 4 semanas, se propone agregar al componente todas las funcionalidades restantes relacionadas con el cambio de formato de sus elementos, la generación de gráficos, la agrupación de datos, entre otras; que contribuyen a un mejor funcionamiento de los reportes y al aumento de su flexibilidad (figura 2.7).

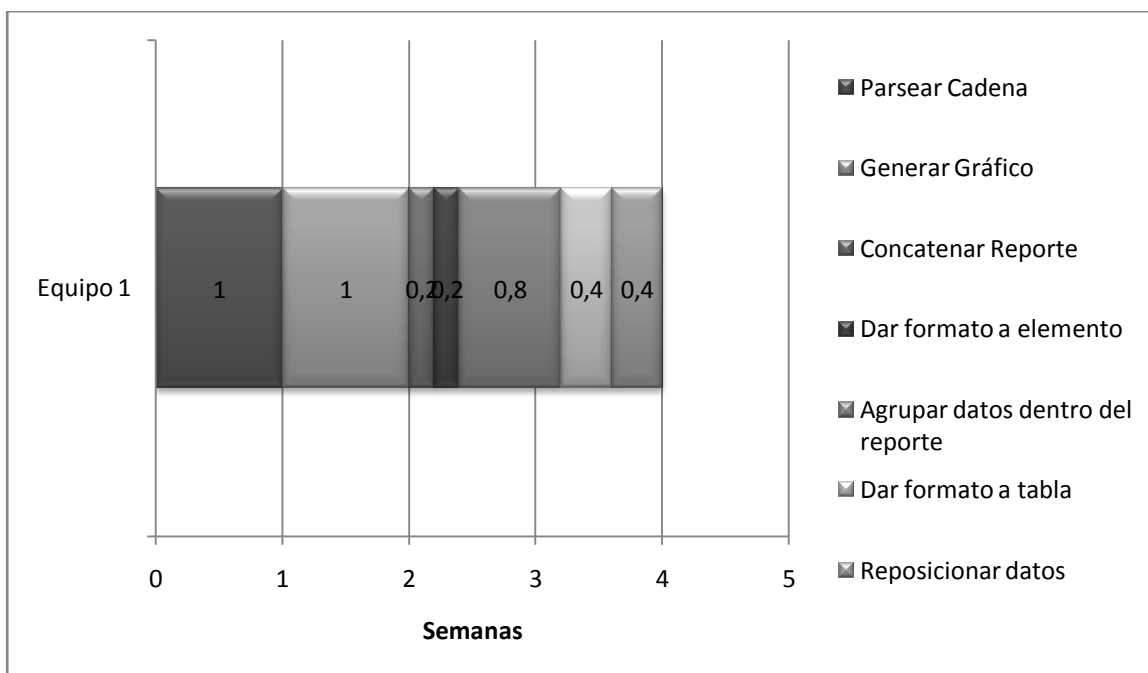


Figura 2.7: Segunda Iteración

El plan de entregas se presenta en la siguiente tabla:

Historia de Usuario	Tiempo Estimado (días)	Iteración Asignada	Entrega Asignada
Generar Reporte	5	1	1
Embeber reporte en un formato determinado	5	1	1
Parsear Cadena	5	2	2
Exportar reporte a formato PDF	2	1	1
Exportar reporte a formato XLS	2	1	1
Generar Gráfico	5	2	2
Cargar Imagen	1	1	1
Concatenar Reporte	1	2	2
Dar formato a elemento	1	2	2
Agrupar datos dentro del reporte	4	2	2
Dar formato a tabla	2	2	2
Reposicionar datos	2	2	2

Tabla 2.1: Plan de entregas

2.3. CONCLUSIONES

En el presente capítulo se analizaron las funcionalidades a incluir en el componente a través de las historias de usuario y siguiendo el proceso de desarrollo propuesto por XP. Al final del mismo se obtuvo una planificación de dos iteraciones que incluyen las historias de usuario identificadas y que tienen como objetivo dotar a los usuarios de un componente en corto plazo que cumpla con las funcionalidades elementales y que irá evolucionando en iteraciones posteriores.

CAPÍTULO 3. IMPLEMENTACIÓN DE LA SOLUCIÓN

En el capítulo anterior, el proceso de desarrollo fue descompuesto en iteraciones de 3 a 4 semanas ficticias, de 40 horas, práctica de XP basada en el desarrollo de software como un ejercicio creativo y por tanto la necesidad de contar con un equipo fresco y descansado. Además, dicha planificación está basada en la práctica de pequeñas entregas frecuentes. Cada una producirá un conjunto de casos de prueba funcionales para cada una de las historias desarrolladas en la iteración.

El presente capítulo se encarga de exponer el proceso seguido para la construcción del componente arquitectónico haciendo énfasis en sus características esenciales y la forma de dar cumplimiento a las funcionalidades identificadas, así como las restricciones surgidas para garantizar el buen funcionamiento del mismo. Además provee de un conjunto de artefactos obtenidos a raíz de la aplicación de la metodología de desarrollo seleccionada.

3.1. ITERACIONES PARA LA PRODUCCIÓN DE ENTREGABLES

TAREAS DE IMPLEMENTACIÓN

Para desarrollar las historias de usuario pertenecientes a cada una de las iteraciones identificadas se hizo necesario descomponer las mismas en tareas de implementación, las cuales se listan en los Anexos 2 y 3 para las iteraciones 1 y 2 respectivamente.

DISEÑO

Para la construcción del componente se dividió el mismo en paquetes distribuidos de forma tal que cada uno englobe funcionalidades comunes e independientes de los demás paquetes, lo cual provee un diseño con bajo acoplamiento y alta cohesión. (Figura 3.1)

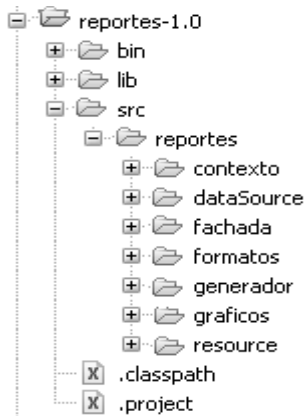


Figura 3.1: Distribución de paquetes en el componente

Los paquetes identificados fueron los siguientes:

contexto: Contiene las clases encargadas de manejar el contexto de JSF tanto para cargar los recursos de la aplicación como para obtener el response donde será escrito el flujo de salida que contiene los datos para mostrar los distintos archivos.

resources: Contiene las clases encargadas de obtener los recursos necesarios para generar los reportes dentro del contexto.

formatos: Contiene las clases encargadas de manipular el objeto JasperDesign para modificar los formatos de los reportes.

graficos: Contiene las clases encargadas de construir los gráficos.

dataSource: Contiene las clases encargadas de manipular el origen de datos en elementos de valor para el reporte.

generador: Contiene las clases encargadas de exportar los reportes a los distintos formatos de salida.

fachada: Funciona como punto de partida para todas las funcionalidades del componente.

Para comenzar a implementar cada una de las tareas se realizó un diseño simple previo, que incluía solamente las clases necesarias a implementar. El mismo se hizo utilizando tarjetas CRC las cuales son usadas para representar las responsabilidades de las clases y su interacción. El nombre CRC viene dado por clase, responsabilidad y colaboración, elementos que sus creadores vieron como

las dimensiones esenciales del modelo orientado a objetos. Las mismas se utilizaron para resumir el significado de las clases y estructurar su conjunto simulando escenarios. Al culminar ambas iteraciones el diseño quedó como se muestra en la figura 3.2.

IMPLEMENTACIÓN

Para implementar las funcionalidades detalladas con anterioridad se hizo necesario intervenir en el ciclo de vida de JasperReports y realizar cambios en los objetos que se iban generando durante su ocurrencia.

El ciclo de vida estudiado en la fase de Exploración comienza a partir de la obtención de un archivo *.jrxml* que contiene el diseño del reporte a generar. Dicho archivo debe encontrarse en el classpath del proyecto, en este caso en el directorio WebContent/resources. La clase utilizada para realizar las acciones de cargar los reportes dentro del componente es JasperReportResource respondiendo a la tarea #1 identificada.

Los reportes no son los únicos recursos que se hace necesario cargar, en el mismo caso se encuentran las imágenes. Las mismas son los elementos gráficos más complejos que se pueden tener en un reporte. Como los campos de texto, su contenido es dinámico y evaluado en tiempo de ejecución usando una expresión.

El valor retornado por dicha expresión es usado como el código para la imagen que será mostrada. Es introducido por el elemento `<imageExpression>` y puede retornar valores en el rango de clases siguiente: `java.lang.String`, `java.io.File`, `java.net.URL`, `java.io.InputStream` o `java.awt.Image`.

Teniendo en cuenta que el contexto de JSF provee los medios para obtener un objeto de tipo URL a partir de una dirección del classpath dada, se generalizó para todo el CICPC utilizar en las imágenes una expresión cuyo valor de retorno sea del tipo `java.net.URL`, el método para cargar las imágenes se encuentra también en la clase JasperReportResource. Dicha funcionalidad responde a la tarea #9 identificada.

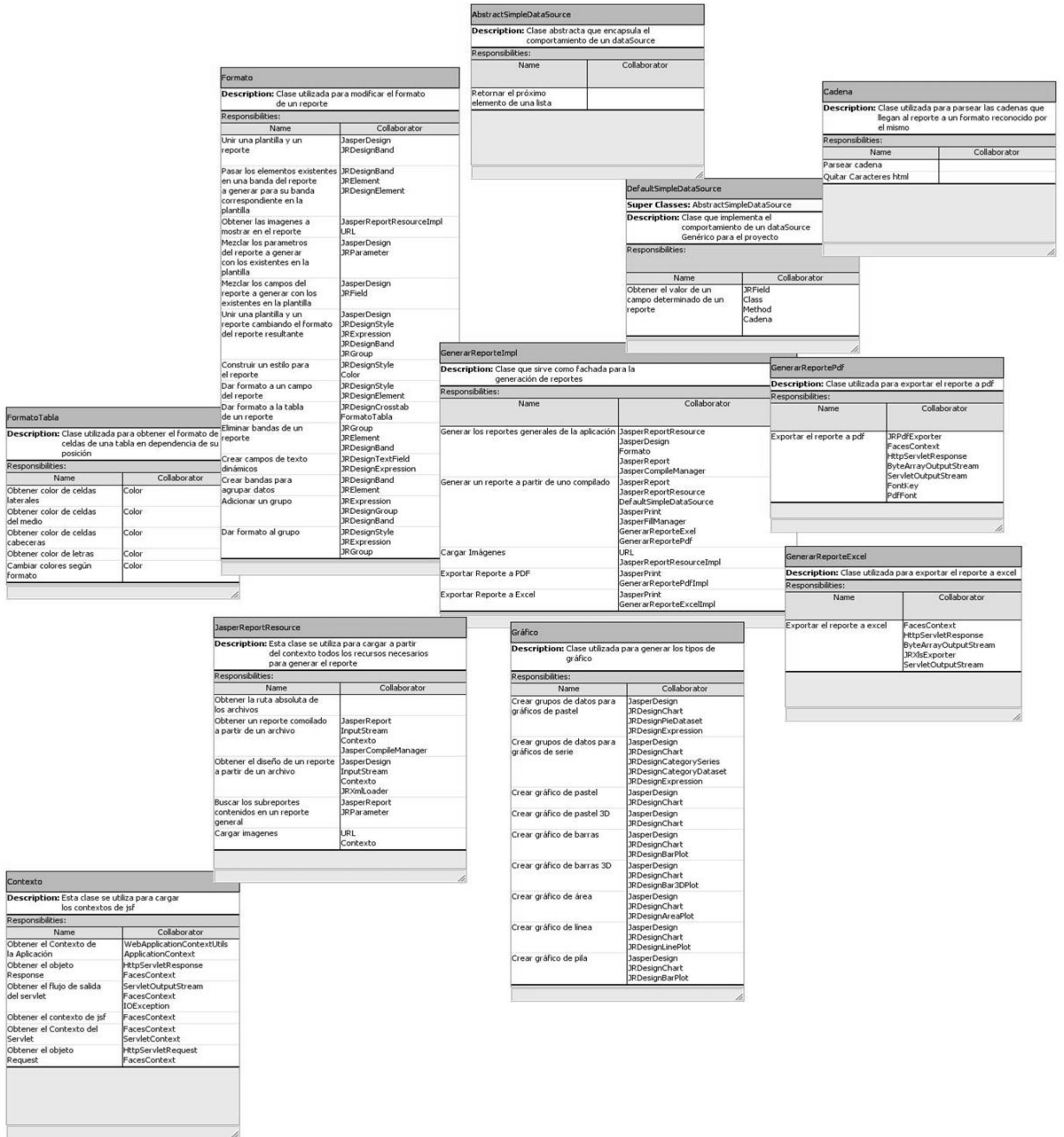


Figura 3.2: Tarjetas CRC

Como es de esperar cada uno de los reportes a generar en el proyecto CICPC tienen su diseño específico, pues muestran datos diferentes y responden a documentos oficiales con características propias. Sin embargo, todos deben seguir una estructura que varía en dependencia del grupo de documentos al que pertenezca, los cuales ya se definieron anteriormente.

Esta estructura generaliza los encabezados y pie de página de cada grupo de reportes, por lo que se hace posible la creación de archivos generales con estos formatos, lo cual posibilita que a la hora de diseñar un reporte, no sea necesario incluir las bandas pageHeader y pageFooter, limitando el diseño solamente a aquellos elementos específicos del reporte, lo cual ahorra tiempo de diseño. Además, un cambio en un elemento determinado de uno de estos formatos, provocaría un cambio solamente en el archivo general; de lo contrario, sería necesario cambiar el archivo de cada reporte perteneciente a dicho grupo.

La construcción de los diseños con estas estructuras responde a la tarea #5. Sin embargo, no sería suficiente con crear los archivos, sería necesario además mezclarlos con los archivos originales pertenecientes al reporte para obtener como resultado final un único reporte con las características de ambos.

El próximo paso consiste, por tanto, en analizar la siguiente etapa en el ciclo de vida del reporte, donde se obtiene el objeto JasperDesign. Como ya se explicó el objeto JasperDesign contiene en su estructura todos los elementos de diseño del reporte en forma de objetos y provee diversas funcionalidades para la modificación de los mismos. Es por esta razón que dicho objeto es de gran utilidad para la implementación de un elevado por ciento de funcionalidades a incluir en el componente.

La principal característica que tiene el proceso de mezclar reporte es la de mezclar sus bandas, para obtener un reporte en el que todas sus secciones coincidan con los datos que se desean mostrar. La clase utilizada para realizar las acciones de mezclar los reportes dentro del componente es Formato, respondiendo a la tarea #6 identificada.

El reporte generado estará vacío si no se insertan elementos dentro de su diseño. Los elementos del reporte son objetos que se muestran como textos estáticos, textos dinámicos, imágenes, líneas o rectángulos; que se insertan dentro de las secciones en el diseño del reporte para que aparezcan en el documento final.

Las propiedades que son comunes a todos los tipos de elementos están agrupadas en el tag <reportElement> y pueden aparecer en la declaración de todos los elementos del reporte. Entre ellas

se encuentran: Posición Absoluta, Posición Relativa, Tamaño y Color, las cuales deben ser tomadas en cuenta a la hora tanto de crear como de modificar los elementos del reporte.

Para construir un texto dinámico se utiliza uno de los métodos implementados en la clase Formato que se encarga de construir un objeto de tipo JRDesignTextField que constituye el diseño de los elementos de texto dinámico a partir de una expresión, posiciones "X" e "Y", además de su tamaño. Dicha funcionalidad tributa a la tarea #20 identificada.

Otra de las funcionalidades que debe brindar el sistema a construir es la de dar formato a un reporte estadístico determinado. Dar formato a un reporte contempla dar formato a cada uno de sus elementos, en este caso objetos de tipo JRTextElement, JRTextField y JRCrosstab; que aunque comparten el padre JRElement y por lo tanto poseen las características anteriormente mencionadas, es necesario darles formato por separado, pues poseen diferencias entre sí.

Para los elementos de texto, dar formato consiste en cambiar su color de fondo, su color de letra, el tamaño de la letra y algunas propiedades en los estilos de la misma. Dichas propiedades están contenidas en un objeto de tipo JRDesignStyle el cual constituye una propiedad de todos los elementos.

Para crear un estilo se utiliza el método de firma JRDesignStyle estilo(Color forecolor, Color backcolor, String nombre) ubicado en la clase Formato, el cual responde a la tarea #18 identificada.

Dar formato a las tablas constituye un proceso más complicado, pues sus atributos se deben cambiar en dependencia de la posición de la celda que se esté modificando, teniendo en cuenta que el encabezado de la tabla, las cabeceras de filas, las cabeceras de columnas, las celdas centrales y los totales son de colores diferentes.

Para implementar dicha funcionalidad se utiliza una clase de apoyo, FormatoTabla, que provee el formato que se debe dar según el tipo de celda que se esté modificando. El método principal se encuentra ubicado a su vez en la clase Formato. La estructura que presentan las tablas en JasperReports es la siguiente:

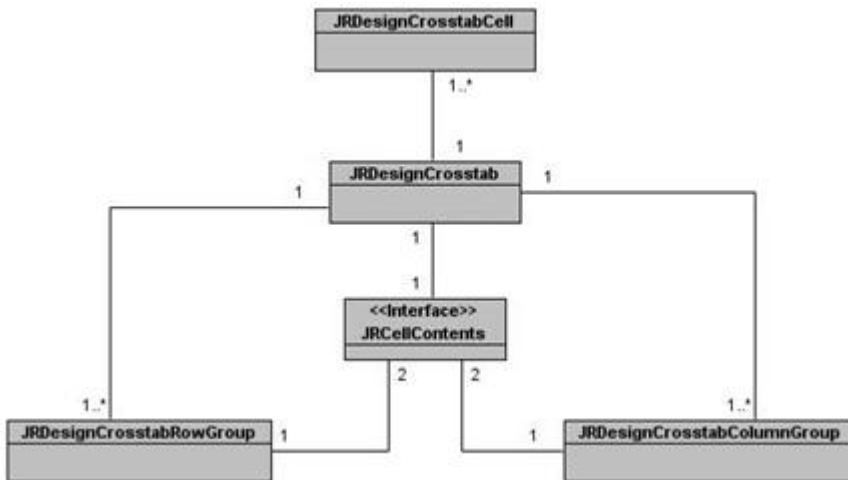


Figura 3.3: Estructura de las tablas en JasperReports

Existen otros elementos que forman parte de dicha estructura pero que su estudio no se hizo necesario para la construcción de esta funcionalidad. El elemento `JRDesignCrosstab` contiene una lista de objetos del tipo `JRDesignCrosstabCell`, los cuales representan las celdas del centro de la tabla. Contiene además una lista de objetos de tipo `JRDesignCrosstabColumnGroup`, los cuales representan las columnas de la tabla y que a su vez contienen dos elementos de tipo `JRCellContents` que representan la cabecera de la columna y el total de la misma. Otro de sus elementos es una lista de objetos de tipo `JRDesignCrosstabRowGroup`, los cuales representan las filas de la tabla y que contienen a su vez una cabecera y un total almacenados en el mismo tipo de datos explicado anteriormente. Finalmente la tabla contiene un elemento de tipo `JRCellContents` que en este caso se refiere a la celda que encabeza la tabla.

Una vez comprendida dicha estructura basta con usar la clase `FormatoTabla` para obtener el formato necesario y darle el mismo a cada uno de los elementos citados anteriormente. Dicha funcionalidad responde a la tarea #22 identificada.

Otra de las funcionalidades requeridas es la de agrupar elementos de forma dinámica en los reportes. El objeto de tipo `JasperDesign` está compuesto por otro de tipo `JRDesignDataSet` que contiene a su vez una lista de objetos de tipo `JRGroup`; este sería el lugar correcto para insertar los grupos deseados.

Los grupos representan una forma flexible de organizar los datos dentro del reporte. Un grupo es representado por una secuencia de campos consecutivos en el datasource que tienen algo en común, como el valor de un campo por ejemplo. Cuenta con tres componentes: una expresión, una cabecera y un pie de sección. El valor de la expresión asociada al grupo es la que realiza la agrupación de los campos, y representa el valor que los mismos tienen en común.

La funcionalidad de construir grupos en tiempo de ejecución significa que se debe construir un objeto de este tipo en dependencia de los datos que se deseen agrupar y que cuente con todas las características antes descritas. Para ello se utiliza un método ubicado también en la clase Formato el cual responde a la tarea #19 identificada.

La banda correspondiente a la cabecera del grupo debe estar encabezada por el valor de la expresión por la que se está agrupando, seguida por los elementos contenidos en la cabecera de columna del reporte sin el que contiene la expresión por la que se agrupa.

Para copiar a la cabecera del grupo los datos de la cabecera de columna del reporte se utiliza el método de firma String crearBandaGrupo(String grupo, JRDesignBand bandaPartida, JRDesignBand bandaResultado, boolean horizontal, List<String> valores, Integer valor), el cual además de copiar los elementos de una banda a otra y eliminar el que será mostrado como expresión del grupo, se encarga de repositionar los elementos dentro del reporte de forma tal que el diseño del mismo se mantenga. Dicha funcionalidad tributa a responder la tarea #24 identificada.

Una vez realizado dicho proceso solamente resta agregar a la banda el elemento de texto con la expresión correspondiente al grupo, para lo cual se utiliza el método JRDesignTextField obtenerTextoDinamico (JRDesignExpression exp, int x, int y, int alto, int ancho). Dichas funcionalidades responden a la tarea #20.

La banda correspondiente al pie del grupo debe contener solamente la cantidad de elementos contenidos en cada grupo, los elementos de texto que muestran dicha información están contenidos en el pie de columna del reporte, por lo cual sólo es necesario crear la banda y agregarle dichos elementos. La construcción de esta funcionalidad responde a la tarea #21.

Aunque después de creadas estas funcionalidades el grupo ya se encuentra adicionado al reporte, es necesario además modificar el detail del mismo para repositionar sus elementos de forma tal que coincidan con los de la cabecera del grupo. Además es necesario modificar la tabla para que sus filas

se orienten a la expresión por la que se agrupa el reporte, por lo cual se cambia la expresión de la misma por la del grupo.

Existen situaciones en las que se hace necesario mostrar u ocultar ciertas bandas según un criterio. Dicho criterio está asociado a los reportes estadísticos y se manifiesta según el tipo de reporte que se desee generar (cualitativo o cuantitativo). Un reporte cualitativo debe mostrar todos los datos diseñados en el archivo XML, mientras que uno de tipo cuantitativo solo debe mostrar los encabezados y pies de página así como la sección del sumario que contendrá los cálculos estadísticos y gráficos necesarios para este tipo de reporte.

Para eliminar toda la información que resulta intrascendente en este caso se utiliza otro método de la clase Formato en el cual la cabecera y pie de la columna principal así como las de los grupos generados y la sección del detail son sustituidas por una banda en blanco que no representa ningún espacio para el reporte. Como ya se había dicho anteriormente la banda del sumario se mantiene, pero solo con las tablas cuantitativas y los gráficos que contenga, por lo cual simplemente se obtienen dichos elementos del sumario actual y se insertan en uno nuevo desde el comienzo de la banda. Es importante destacar que para que el método funcione, la tabla colocada en el proceso de diseño del *.jrxml* debe tener como clave "tabla". Dicha funcionalidad responde a la tarea #23.

Otra funcionalidad necesaria es la generación de gráficos en el reporte. JasperReports soporta varios tipos de gráficos incluyendo de pastel, barras, líneas, entre otros. La estructura que presentan los gráficos en JasperReports es la siguiente:

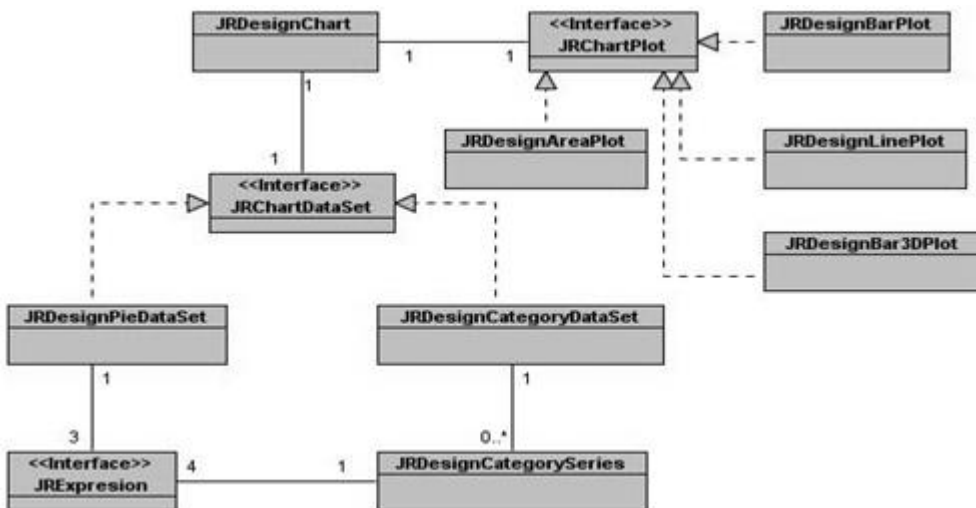


Figura 3.4: Estructura de los gráficos en JasperReports

Existen otros elementos que forman parte de dicha estructura pero que su estudio no se hizo necesario para la construcción de esta funcionalidad. Como se puede observar todos los gráficos parten de un objeto de tipo JRDesignChart, el cual contiene las especificaciones de los datos a mostrar así como su diseño. La forma de especificar los datos a mostrar difiere en cuanto a dos tipos fundamentales de gráficos, los de pastel y los de categoría que incluyen barras, pila, área, entre otros. Existe otro tipo de especificación de datos pero pertenece a tipos de gráfico que no se hace necesario generar. Esta diferencia provoca que se deban construir los gráficos pertenecientes a cada uno de los tipos por separado.

Sobre la interfaz JRChartPlot se puede concluir que es implementada por varias clases para cada tipo de gráfico, por lo cual las modificaciones en las propiedades contenidas en los objetos de este tipo deben realizarse individualmente. El estudio del diseño anterior responde a la tarea #14.

Gráficos de Pastel

JasperReports permite crear gráficos de pastel en dos y tres dimensiones. El procedimiento para construirlos es idéntico. Es por esta razón que dentro de la clase Gráfico del componente se implementó un método que generalizó dicho procedimiento, el cual construye un gráfico a partir de las expresiones de clave y valor recibidas además de ajustar algunas propiedades como su tamaño y posición. Dicha funcionalidad responde a la tarea #15.

Gráficos de Barras

Como los gráficos de pastel; los gráficos de barras pueden ser usados para ilustrar diferencias cuantitativas entre los elementos del gráfico. Una de las ventajas que tienen es que puede mostrar los mismos datos para más de una categoría.

Los demás tipos de grafico son los de pila, de línea y de área; además de los correspondientes en 3D. Aunque el procedimiento para construirlos no es exactamente idéntico, se pudo generalizar la construcción de su dataset y el suministro de algunas propiedades mediante la realización de un proceso que se centra en la modificación del objeto JRChartPlot el cual tiene una implementación diferente para cada tipo de gráfico y contiene los atributos de orientación así como las diferentes propiedades que intervienen en el diseño como mostrar etiquetas, títulos, entre otras. Dicha funcionalidad responde a la tarea #16.

Las funcionalidades explicadas son las que hacen uso del objeto JasperDesign para su implementación y por lo tanto se insertan en el ciclo de vida del reporte después de la creación del mismo. Una vez realizadas estas funcionalidades en correspondencia con los requerimientos del usuario, se procede a compilar el reporte para crear el objeto de tipo JasperReport.

Entre los elementos que contienen los reportes se encuentran los subreportes, los cuales son una importante funcionalidad de las herramientas generadoras de reportes. Permiten crear reportes más complejos y simplifican el trabajo de diseño. Un subreporte es un reporte normal que es incorporado como parte de otro reporte. Es además una forma especial de elementos del reporte.

Como en un reporte normal, el diseño es un objeto JasperReport obtenido después del proceso de compilación. El elemento subreporte tiene una expresión que es evaluada en tiempo de ejecución para obtener el código del objeto JasperReport a cargar. Dicha expresión puede retornar valores de las siguientes clases: `java.lang.String`, `java.io.File`, `java.net.URL`, `java.io.InputStream` y `net.sf.jasperreports.engine.JasperReport`.

En el proyecto se decidió generalizar el uso de expresiones con valores de retorno de tipo JasperReport declaradas como parámetro, pues el proceso de insertar subreportes se reduciría a cargar el mismo y pasárselo al reporte maestro dentro de un mapa de parámetros.

Teniendo en cuenta esta generalidad se implementó el método de firma `List<String> buscar(JasperReport reporte, String prefijo)`, ubicado en la clase `JasperReportResource`, el cual se encarga de buscar dentro del reporte los subreportes que se desean cargar y proveer una lista con las direcciones donde los mismos se encuentran a partir de la unión del prefijo que indica la subcarpeta del subreporte, que coincide con la del reporte maestro; así como el nombre del parámetro declarado para ser utilizado en la expresión del subreporte, el cual debe coincidir con el nombre del subreporte a cargar. Dicha lista es utilizada posteriormente para cargar los reportes respondiendo a la tarea #2.

Una vez obtenidos todos estos datos ya se está en condiciones de realizar el llenado del reporte, este proceso se realiza haciendo uso de la clase `JasperFillManager` la cual necesita el objeto `JasperReport` con el reporte a generar, un mapa de parámetros que contenga los valores de los parámetros declarados en el reporte y un `datasource`.

JasperReports soporta varios tipos de `datasource` usando una interfaz especial llamada `JRDataSource`. Se puede usar cualquier tipo de `datasource`, con solo implementar la interfaz antes

mencionada, o usando una de las implementaciones que provee JasperReports para realizar wraps de colecciones o arreglos de JavaBeans, CSV o archivos XML, entre otros.

La interfaz JRDataSource propone la implementación de 2 métodos, public boolean next() y public abstract Object getFieldValue(JRField field). El primero de los métodos se debe encargar de obtener el próximo elemento en el origen de datos suministrado, mientras que el segundo se debe encargar de buscar el valor del campo que se esté llenando dentro del elemento suministrado por el método anterior.

El datasource implementado en el componente se mueve a través de una lista de datos suministrada, en la cual se marca el elemento a partir de un índice. Dicha lista responde a objetos de clases dentro del dominio de CICPC y por lo tanto la obtención de los valores de sus elementos se debe implementar de forma genérica.

Teniendo en cuenta que los campos del reporte van a responder a los atributos de la clase que se esté pasando como datasource, entonces se debe crear un mecanismo que haga un llamado al get de dicho atributo basándose en el nombre del mismo. Esto es posible hacerlo a partir del uso de Reflection.

Todos los objetos en java heredan de la clase java.lang.Object y por ello están dotados de un método getClass(). Este método devuelve un objeto java.lang.Class, que va a ser el punto de entrada al API Reflection. Entre los métodos con que cuenta dicha clase se encuentra el siguiente:

java.lang.reflect.Method getMethod(String name, Class[] parameterTypes): Devuelve un método público de la clase, a partir de su nombre, y de un arreglo con las clases de los parámetros del método. Si la clase no contiene ningún método con ese nombre y esos parámetros, se lanzará la excepción java.lang.NoSuchMethodException.

Luego de obtener el objeto de tipo Method, ya se está en condiciones de invocar al método que devuelve el valor del campo que se desea a partir de la siguiente funcionalidad:

public Object invoke(Object obj, Object[] args): Ejecuta el método sobre un objeto, pasándole los parámetros necesarios, y devuelve su resultado. Dicha funcionalidad responde a la tarea #3.

Los campos del reporte representan la única forma de mapear los datos provenientes del origen de datos dentro de las rutinas de generación de reportes los cuales representan los atributos de la clase proveniente en el origen de datos.

Con un basamento en la implementación del datasource genérico, y teniendo en cuenta que el proceso de llenado de los campos se realiza a través de un llamado al método `getFieldValue()` de dicho datasource para cada uno de los campos existentes en el reporte, se evidencia que dichos campos no tienen que ser solamente aquellos que mapeen los atributos de la clase, sino que su obtención se realiza a través del método `getXX()`; lo cual significa que si en la clase se encuentra implementado un método `getXX()` que no corresponda a ningún atributo de la misma, se puede crear un campo que mapee dicho método cuyo nombre será el del método quitándole el prefijo `get` y poniendo la primera letra resultante en minúsculas.

Cuando se declaren los campos, se debe tener la seguridad de que el origen de datos suministrará en el tiempo de llenado del reporte los valores para todos esos campos, pues si un campo es declarado y no tiene el correspondiente `get` en la clase utilizada como origen de datos, se lanzará una excepción en tiempo de ejecución; aunque si alguno de los atributos de la clase no tiene su correspondiente campo en el reporte, esto no afecta las operaciones de llenado, solo que no será accesible.

Luego de obtenido el valor para un campo determinado se chequea su tipo por las siguientes razones:

- Si es un objeto de tipo `Set`, `HashSet` o `PersistentSet`, se convierte en una lista para poder ser manipulada correctamente por la librería.
- Si es un objeto de tipo `String` se debe analizar la cadena para garantizar que la que llegue al reporte cuente con el formato adecuado para su entendimiento.

El análisis de la cadena se centra en tres pasos fundamentales:

- Quitar de la cadena los caracteres HTML no reconocidos por la librería de JasperReports.
- Proveer la fuente necesaria para la generación del archivo PDF a partir del texto a mostrar.
- Cambiar el tamaño de fuente en correspondencia con la escala utilizada por JasperReports.

Los caracteres reconocidos por JasperReports son aquellos correspondientes al HTML básico, la codificación de los acentos y los caracteres extraños no son válidos para el mismo. La siguiente tabla muestra un listado con los caracteres no reconocidos y su correspondiente caracter en JasperReports.

Caracter no Reconocido	Carácter Reconocido	Caracter no Reconocido	Carácter Reconocido
<p>		¡	¡
</p>	\n	­	(espacio)
		¯	-
		¿	¿
		 	\n
		&aOacute;	Ó
á	á	&aUacute;	Ú
é	é	Ñ	Ñ
í	í	ñ	ñ
ó	ó	ü	ü
ú	ú	Ü	Ü
&aAacute;	Á	 	
&aEacute;	É	 	\n
&alacute;	í	"	"

Tabla 3.1: Correspondencia de caracteres en JasperReports

El estudio anterior cumple con la tarea #10 identificada.

Una vez identificados estos caracteres se procede a eliminarlos de las cadenas encontradas durante el llenado del reporte. Para ello se utiliza un método ubicado en la clase Cadena de firma `String quitarCaracteresHtml(String cadena)`; el cual sustituye cada caracter por el correspondiente reconocido. La funcionalidad anterior cumple con la tarea #11 identificada.

Para proveer la fuente necesaria para la generación del PDF a partir del texto a mostrar se utilizaron las clases `FontKey` y `PdfFont` que provee la librería para construir una correspondencia entre ambas propiedades de los elementos. Dicha funcionalidad se implementa durante el proceso de exportación.

El tamaño de fuente de los elementos de texto es medido en puntos y es especificado usando el atributo `size`. Teniendo en cuenta que la escala no se corresponde con la codificación suministrada a los elementos de texto, se hace una conversión de escala en el método de firma `String cadenaReportes(String cadena)`, el cual se encuentra en la clase `Cadena`. Dicha funcionalidad responde a la tarea #13 identificada.

Una vez culminado este proceso se obtiene un objeto de tipo JasperPrint con el reporte listo para imprimir.

Entre las funcionalidades que brinda la librería JasperReports se encuentra la de exportar a varios formatos como PDF, XLS, entre otros.

Todos los datos de entrada que el exportador necesita pueden ser suministrados a través de los llamados parámetros del exportador, antes de comenzar con el proceso de exportación. Esto es porque dicho proceso es invocado llamando al método `exportReport()` que no recibe ningún parámetro por sí mismo, sino que deben ser pasados usando el método `setParameter()`, de la instancia con la que se esté trabajando.

No importa el tipo de salida que se vaya a producir, los parámetros deben ser usados para indicar el lugar al cual se debe enviar dicha salida y el objeto JasperPrint donde se encuentra el reporte. Se pueden agregar otros parámetros que especifiquen propiedades que se deseen en el archivo a generar.

Para exportar a PDF se utiliza la clase `GenerarReportePdf` la se encarga de suministrar al exportador los parámetros necesarios que en este caso coinciden con los generales. Dicha funcionalidad responde a las tareas #7 y #17.

Cuando un reporte se exporta a PDF, la librería JasperReports usa la librería iText. Como su nombre lo indica (Portable Document Format), el archivo PDF puede ser visualizado en varias plataformas y siempre se verá igual. Esto ocurre parcialmente porque en su formato hay una forma especial de trabajar con las fuentes. Si se están diseñando reportes que luego serán exportados a PDF, se debe estar seguro que se seleccionen las fuentes apropiadas. El nombre de fuente introducido en el atributo `fontName` no es usado al exportar a PDF. Existe un atributo especial para especificar las propiedades de fuente que la librería iText espera, el cual es `pdfFontName`.

Teniendo en cuenta que los estilos de los textos no se generan sólo para exportarlos al PDF, se implementó una forma de convertir los estilos provenientes de cualquier cadena en estilos de valor para la librería iText. Para ello se incluyó un mapa de parámetros donde la clave pertenece al estilo de texto proveniente del reporte y el valor al estilo correspondiente en el PDF. Dicho mapa se incluyó como un parámetro más del exportador respondiendo a la tarea #12.

Para exportar a XLS se utiliza la clase `GenerarReporteExcel` la cual contiene un método de firma `void generarReportExcel()` que realiza las mismas funcionalidades del método anterior pero esta vez orientado a imprimir un archivo de tipo Excel. Dentro del método además se adicionan un grupo de parámetros al exportador para obtener un archivo con éxito, entre los mismos se encuentran: que se impriman todas las páginas del reporte en la misma hoja de cálculo, que se detecte el tipo de la celda automáticamente, el nombre de la hoja de cálculo, que el fondo de la página no sea blanco y que los espacios en blanco entre las celdas sean quitados. Dicha funcionalidad responde a la tarea #8.

Finalmente todo el ciclo de vida de los reportes así como las intervenciones realizadas en el mismo por el componente se resumen en la figura 3.5.

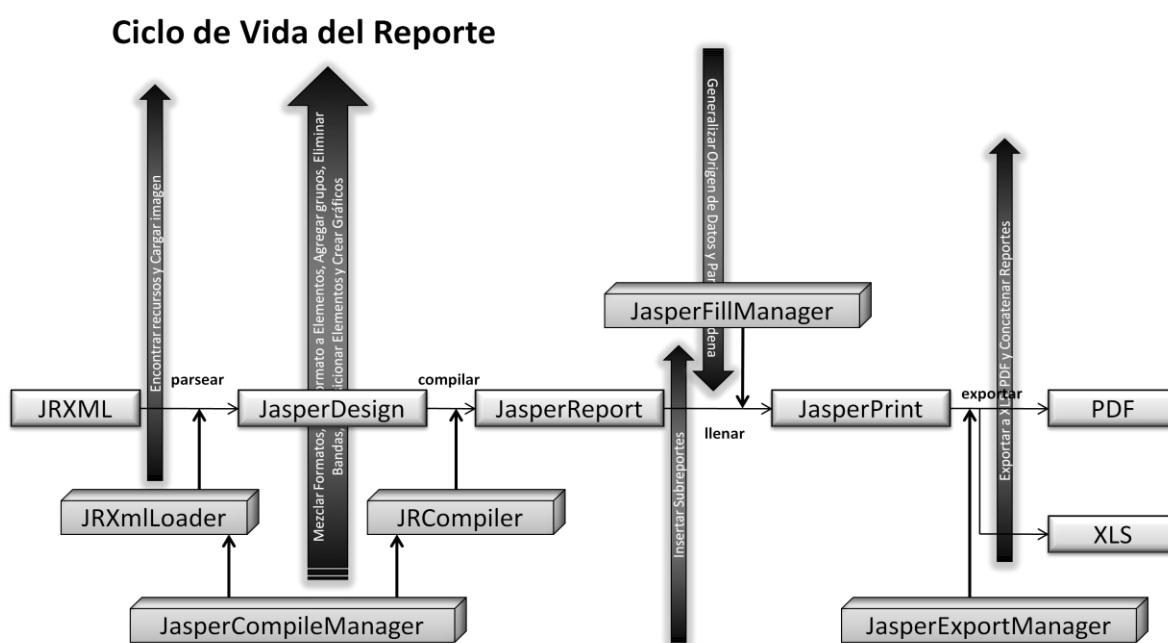


Figura 3.5: Ciclo de vida de JasperReports interceptado por el Componente de Reportes

Analizando las similitudes de las distintas necesidades de generar los reportes en el CICPC, dicho flujo fue generalizado en cuatro funcionalidades:

`public void generarReportes(String reporte, List<?> lista, Map<String, Object> parametros, String form, String tipo) throws JRException, MalformedURLException;` es la funcionalidad utilizada por la mayoría de los módulos del CICPC para generar los reportes, generaliza el flujo básico para generar los reportes que comprende además del ciclo de vida de los mismos las funcionalidades identificadas a

implementar en la primera iteración. Con la utilización de la misma basta con especificar el nombre del reporte, la lista de datos a generar, los parámetros con datos a mostrar en el reporte, el formato que se desea aplicar al mismo y el tipo de salida; para que el componente se encargue de realizar el resto del trabajo y mostrar el reporte requerido por pantalla.

public JasperPrint generarReporte(String reporte, List<?> lista, Map<String, Object> parametros, String form, String tipo) throws JRException, MalformedURLException: Realiza la misma función que el método anterior pero solo llega a la parte del flujo de reportes donde se genera el JasperPrint, el cual es devuelto para ser procesado de forma diferente.

public JasperPrint generarReportesPersonalizados(String reporte, List<?> lista, Map<String, Object> parametros, String banner, Color color, String formato, Integer tamnoLetra, String tipoGrafico, List<String> serie, List<String> clave, List<String> valor, String grupo, String horizontal, Boolean cuantitativamente, String form, int numero, List<String> valores) throws JRException, MalformedURLException: Esta es una funcionalidad más compleja que incluye el uso de todas las funcionalidades del componente. Además de los datos especificados con anterioridad, se hace necesario suministrarle todos los datos correspondientes al diseño en que se desee mostrar el reporte. Estos reportes solo utilizan una porción de su diseño para luego ser rediseñados por el componente ofreciendo mayor flexibilidad. Los que utilizan esta funcionalidad son los reportes del módulo de estadísticas, el cual se encuentra especializado en la gestión de datos y generación de reportes de acuerdo con los requerimientos del usuario.

public JasperPrint generarReportesDataSource(String reporte, AbstractSimpleDataSource dSource, Map<String, Object> parametros, String banner, Color color, String formato, Integer tamnoLetra, String tipoGrafico, List<String> serie, List<String> clave, List<String> valor, String grupo, String horizontal, Boolean cuantitativamente, String form, List<String> valores) throws JRException, MalformedURLException: Este método tiene la misma funcionalidad que el anterior, solo cambia la forma de pasar el datasource, pues anteriormente simplemente se pasaba una lista de elementos y en este caso se pasa un datasource construido con anterioridad. Esta funcionalidad es utilizada en los casos en los que el datasource genérico implementado no se puede aplicar al reporte a generar. Dichas funcionalidades responden a la tarea #4.

PRUEBAS DE UNIDAD

Las pruebas de unidad fueron escritas en varios momentos del desarrollo, como respuesta a necesidades concretas relacionadas con el riesgo de la implementación.

Una vez concluida la primera iteración, el componente comenzó a usarse por el equipo de desarrollo del proyecto CICPC. La interacción de los usuarios con el mismo arrojó una serie de problemas que hacían necesario la realización de una refactorización. Como no se tenía seguridad de cómo se comportaría el componente una vez que se comenzaran a introducir cambios en el mismo, se escribieron pruebas de unidad con el fin de probar las funcionalidades implementadas constantemente para la detección inmediata de posibles fallas. De este proceso surgió la implementación del método testgenerarReportePDF() el cual debía imprimir por pantalla en todo momento un reporte.

Dichas pruebas fueron implementadas con el framework JUnit cargando los contextos de la aplicación haciendo uso de la clase CargadorContextos perteneciente al paquete de pruebas del núcleo del proyecto CICPC, lo cual significa que siempre se estaría probando en un entorno real. Sin embargo, muchas de las funcionalidades utilizadas por el componente no funcionaban correctamente debido a que los contextos de JSF durante las pruebas no eran cargados, por lo cual se hizo necesario la comprobación de que el contexto fuera nulo para tratar la generación de los reportes de forma diferente. Dicho tratamiento incluye cargar de forma estática los recursos y presentar el reporte final al usuario mediante el visor que incluye la herramienta iReport. El método de prueba implementado incluye todo el ciclo de vida del reporte, por lo cual cualquier error proveniente de alguno de los pasos por los que pasa es capturado en el mismo.

Con el comienzo de la segunda iteración se agregaron funcionalidades a la interfaz del componente que, aunque sus características estaban bien definidas, se preveía que la implementación sería complicada. Por esta razón se escribió una nueva prueba con el fin de chequear constantemente el avance exitoso de la funcionalidad, la cual fue testgenerarReporteDesign(), donde se crearon un grupo de variables con los datos a comprobar como formatos, gráficos, entre otros; los cuales pueden ser modificados con el fin de comprobar el total funcionamiento del componente.

En el momento que alguna de estas pruebas fallaba no existía trabajo más importante que resolver el problema, centrándose siempre el equipo de desarrollo en la necesidad de mantener el componente 100% funcional.

3.2. PRODUCTIZACIÓN

La fase de productización requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. (23)

La implementación del componente se ha llevado a cabo dentro del entorno real donde será utilizado el mismo, por lo cual las pruebas de unidad realizadas al código se han realizado con datos reales y con el funcionamiento del resto del proyecto incluido. Es por esta razón que se pudieron detectar los fallos de integración y subsanar los mismos con anterioridad y una vez terminada cada iteración no fue necesario realizar pruebas adicionales.

3.3. CONCLUSIONES

En el presente capítulo se analizaron los procedimientos principales asociados a la implementación del componente y al cumplimiento de todas las funcionalidades previstas. Quedó expuesta la forma de intervenir en el ciclo de vida para la generación de reportes y manipular los distintos objetos generados en las fases para la obtención de los resultados esperados. La culminación del mismo arrojó un componente estable e integrado al entorno productivo. Además de dotar al equipo de desarrollo de una visión más amplia sobre el papel de un componente para la generación de reportes dentro de una aplicación de gestión.

CAPÍTULO 4. VALIDACIÓN DE LA SOLUCIÓN

En el presente capítulo se verifican de manera unitaria e integrada todos los elementos del componente de software y se realiza junto a los usuarios finales la validación de la solución desarrollada a fin de contar con todas las evidencias que garanticen el cumplimiento en alcance, funcionalidad y calidad que el cliente espera.

Además se realiza un estudio de los aspectos favorables y desfavorables que tuvieron lugar durante el desarrollo de la solución y el impacto social de la misma haciendo énfasis en los aportes alcanzados sobre los procesos para la generación de reportes y las herramientas existentes para llevarlos a cabo.

4.1. MANTENIMIENTO

El mantenimiento es un estado normal en un proyecto XP pues se deben producir nuevas funcionalidades y mantener el sistema corriendo simultáneamente. Durante esta fase se intentan grandes refactorizaciones no realizadas en la iteración culminada, se insertan nuevas tecnologías o se migra a versiones más actuales de las que ya se están usando, se experimentan nuevas ideas arquitectónicas y el cliente puede escribir nuevas historias en pos de buscar mejoras para su empresa.

Durante la fase de mantenimiento fueron detectadas diversas fallas en la implementación del componente, arrojadas a partir del desarrollo de pruebas funcionales por el cliente; las cuales se fueron mitigando durante la refactorización llevada a cabo. Entre las mismas se encontró que solamente se parseaban las cadenas pertenecientes a los campos del reporte, obviándose por lo tanto aquellas pasadas como parámetros. Por esta razón se agregó una funcionalidad a la clase Cadena que una vez recibido un mapa de parámetros los parseara de la misma forma que los campos; la cual se usó previamente al llenado del reporte.

Además, muchas veces, los tag HTML `<p>`, `` y `
` traían inmersas otras cadenas por lo cual no eran eliminados y por lo tanto JasperReports no los reconocía. Para resolver este problema se trataron como expresiones regulares que abarcaran todas las formas en que pudieran presentarse. Las mismas fueron: `<p[^\>]*>`, `<span[^\>]*>` y `<br[^\>]*/>`.

También se detectó que en el momento de dar formato a los reportes estadísticos no se tenían en cuenta los subreportes a generar, por lo cual se agregó una funcionalidad en la clase Formato

encargada de dar el formato requerido a un subreporte además de modificar la forma de cargar los mismos para permitir la realización de dicho proceso.

Además se encontraron subreportes que podían utilizarse por distintos módulos en el proyecto, y debido a la forma en que se encontraba implementada la obtención de los mismos era necesario duplicarlos para cada subcarpeta del directorio donde se encontraba el reporte maestro que lo incluiría. Por esta razón se implementó en la clase JasperReportResource un método recursivo que es llamado en caso de que un reporte no se encuentre en el directorio especificado. De ser así, se comienza a buscar en todas las subcarpetas del directorio WebContent/resources hasta su tercer nivel que corresponde a los submódulos del proyecto, de esta forma se pueden encontrar los recursos ubicados en cualquier lugar del directorio. Con esta nueva funcionalidad ya no sería necesario especificar la subcarpeta a la que pertenece cada reporte, sin embargo, es preferible no abusar de esta funcionalidad debido al tiempo de ejecución que requiere.

Una vez analizadas las necesidades de la segunda iteración también se detectó que las librerías utilizadas hasta ese momento no cumplían con todas las funcionalidades necesarias y, sin embargo, ya existían nuevas versiones de las mismas que sí las incluían. Por este motivo se migró de jasperreports-1.3.3 a jasperreports-3.1.0 y de jfreechart-1.0.0 a jfreechart-1.0.12. Con esta modificación se garantizaba el cambio de codificación de las cadenas al exportar a PDF y una mayor calidad visual para los gráficos. La herramienta iReport no se pudo sustituir porque las versiones posteriores modificaban algunas propiedades en los elementos que no eran compatibles con los diseños ya creados y se podría crear inconsistencia en los mismos una vez que se intentaran cargar con una nueva versión.

El cliente a su vez identificó la necesidad de una nueva funcionalidad a implementar una vez entregada y puesta en práctica la segunda iteración. La misma se refiere a una nueva forma de tratar las plantillas.

Historia de Usuario	
Número: 13	Usuario: Desarrollador de CICPC
Nombre historia: Modificar Plantillas	
Prioridad en negocio: Baja	Riesgo en desarrollo: Alto
Puntos estimados: 5	Iteración asignada: 3
Descripción: Modificar en las plantillas de los reportes las imágenes iniciales y	

finales y el texto inicial. Una vez que dichos datos sean modificados deben tenerse en cuenta para todos los reportes generados a partir de ese momento.

Observaciones: Debe cambiar la forma de guardar las imágenes principales así como convertir el texto estático principal en un texto dinámico.

Como se puede observar, la historia anterior cambia la concepción de la generación de los reportes, haciéndose necesario dinamizar las plantillas aplicables a los distintos formatos de los reportes. Es por esta razón que se decidió posponer su implementación para una iteración futura que se corresponde con un nuevo ciclo de desarrollo del proyecto CICPC.

4.2. VISIÓN GLOBAL DEL TRABAJO REALIZADO

Con la culminación y puesta en práctica del componente arquitectónico para la generación de reportes en el proyecto CICPC se garantiza la creación de una aplicación menos expuesta a errores, pues se logra la centralización del código referente a los reportes realizándose la mayoría de los cambios dentro del componente y posibilitándose por tanto una manera más eficiente de probar dichos cambios. Además existe una delimitación de responsabilidades en las clases del componente, lo cual enfoca a los desarrolladores en los errores que se puedan producir pues su tipo depende de la clase, y por lo tanto el proceso en el que tuvo lugar.

AHORRO EN TIEMPO DE DESARROLLO

El ahorro del tiempo de desarrollo fue calculado a partir de las siguientes variables:

En primer lugar se calculó un promedio optimista en cuanto al tiempo necesario para incluir los elementos pertenecientes a las plantillas generales en cada reporte en particular. El resultado fue de 5 minutos distribuido en el tiempo para adicionar los elementos e incluir los parámetros que sustentan los mismos.

Teniendo en cuenta que después de culminada la segunda iteración del proyecto CICPC ya se contaba con 812 reportes, el tiempo ahorrado en dicha tarea alcanza un valor de 68 horas hombre que viene dado por la ecuación:

Cantidad de diseños * 5 min

Posteriormente se calculó el tiempo empleado en realizar el proceso para la generación de reportes directamente con la librería de JasperReports. El resultado fue de 25 minutos distribuidos en 15 minutos para la creación de un origen de datos, 5 minutos para la obtención de subreportes y los 5 minutos restantes para cubrir los pasos necesarios para llevar a cabo el ciclo de vida de los reportes.

Teniendo en cuenta de que una vez culminada la segunda iteración ya existían 224 llamadas a la fachada del componente sin contar con las realizadas desde el módulo de estadísticas que por su complejidad será tratado de forma diferente, el tiempo ahorrado para dicha tarea alcanza un valor de 94 horas hombre que viene dado por la ecuación:

Cantidad de llamadas (excluyendo estadísticas) * 25 min

En cuanto al tiempo empleado en realizar el proceso para la generación de reportes directamente con la librería de JasperReports desde el módulo estadísticas, el resultado fue de 90 minutos distribuidos en 15 minutos para la creación de un origen de datos, 10 minutos para la creación de los gráficos, 60 minutos para la manipulación del diseño y los 5 minutos restantes para cubrir los pasos necesarios para llevar a cabo el ciclo de vida de los reportes.

Después de culminada la segunda iteración existían 18 llamadas al componente desde el módulo estadísticas, el tiempo ahorrado para dicha tarea alcanza un valor de 27 horas hombre y viene dado por la ecuación:

Cantidad de llamadas (estadísticas) * 90 min

La suma de todos estos tiempos tiene un valor de 189 horas hombre ahorradas en el tiempo de desarrollo. Sin contar el tiempo necesario para adquirir los conocimientos y la experiencia del trabajo con la librería, teniendo en cuenta que a los programadores del proyecto CICPC, con la utilización del componente, les basta con el estudio de un tutorial realizado con el fin de explicar la forma de utilizar el mismo para ser capaces de realizar sus tareas. Esto significa que aún cuando se está implicado en el proceso de generación de reportes el programador trabaja de forma independiente a la librería JasperReports y por lo tanto se puede abstraer completamente de la misma.

AHORRO DE LÍNEAS DE CÓDIGO

En cuanto al ahorro de líneas de código, para generar un reporte cualquiera, sin contar con el datasource y el código relacionado con gráficos y formatos, es necesario escribir 10 líneas de código como mínimo. Teniendo en cuenta de que en todo el proyecto se realizan 242 llamadas al componente, se ahorraron 2420 líneas de código dadas por la ecuación:

Cantidad de llamadas * 10 loc

4.3. ANÁLISIS DE FACTIBILIDAD

Una vez integrado el componente en el entorno de desarrollo de CICPC, el mismo comenzó a utilizarse por los distintos módulos del proyecto conteniendo todas las funcionalidades que los mismos necesitaban para los procesos de generación de reportes.

Para la utilización del componente en los módulos no especializados en la generación de informes; basta con la realización de un diseño del reporte con la herramienta iReport. Dicho diseño debe cumplir con restricciones documentadas en una guía elaborada con el fin de enfocar a los desarrolladores en la realización de un diseño uniforme y a la vez suministrarles los conocimientos mínimos necesarios para el trabajo con la herramienta. Dicha guía se encuentra ubicada en el repositorio del proyecto y es de total acceso para todos los integrantes del mismo.

Posteriormente a la realización del diseño, el código necesario para obtener el servicio principal que brinda el componente, se encuentra ubicado en las páginas JSF de la aplicación como parte del botón que inicia la funcionalidad de generación de reportes Imprimir/Exportar a PDF.

Un ejemplo de dicho componente es el siguiente:

```
<h:commandButton id="bImprimir" value="Imprimir/Exportar" styleClass="boton long">
    <f:actionListener type="vnz.cicpc.comun.web.util.ReportGeneratorActionListener" />
    <f:param name="reportURL" value="registrocontrol/ver detalles trazas" />
    <f:param name="reportData" value="#{verAuditoriaManejado.object}" />
    <f:param name="reportFormat" value="PDF" />
    <f:param name="reportTemplate" value="formatoConsultar" />
    <f:param name="reportParameters" value="#{verAuditoriaManejado.parametrosReporte}" />
</h:commandButton>
```

El parametro reportURL representa el directorio dentro de WebContent/resources donde se encuentra el diseño del reporte previamente elaborado, reportData representa la lista de datos que se desea mostrar en el reporte, reportFormat el formato de salida en que se mostrará el reporte, reportTemplate la plantilla que se adicionará al diseño creado en dependencia del grupo al que pertenece el reporte y reportParameters los parametros que se desee incluir en el proceso de llenado del reporte.

Esta forma de usar el componente fue definida por el equipo de arquitectura del proyecto e implementada por el mismo. Está compuesta de varios componentes pequeños organizados según el diagrama representado en la figura 4.1:

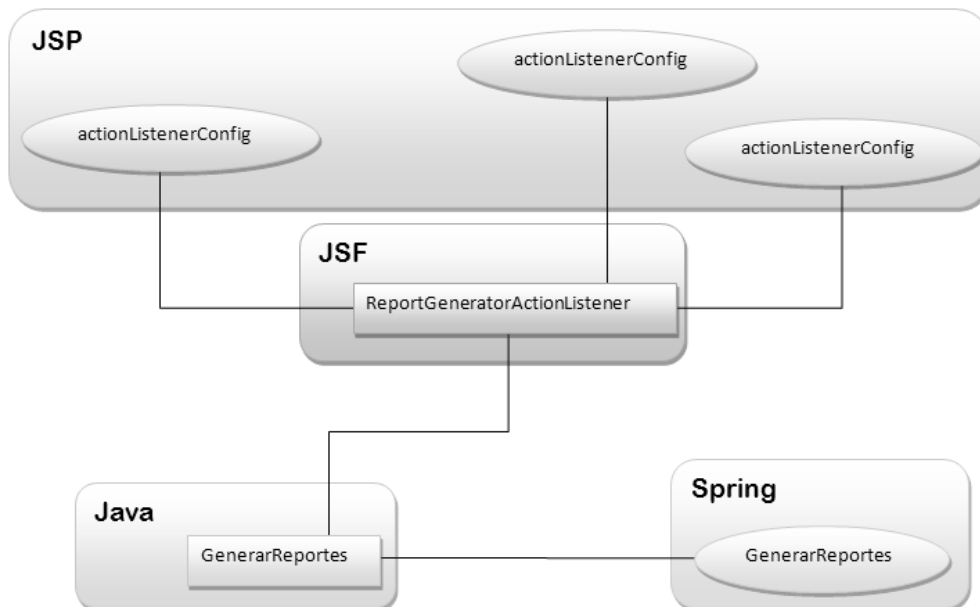


Figura 4.1 : Manejo de los reportes en CICPC

Donde ReportGeneratorActionListener, GenerarReportes y su configuración correspondiente en Spring; son elementos arquitectónicos inmutables que sirven de soporte para los actionListenerConfig a desarrollar, los cuales contienen el conjunto de parámetros asignados al componente. Esta explicación se encuentra mejor detallada en un documento generado por el equipo de arquitectura donde se explican los puntos necesarios para hacer uso de dicha implementación y que se encuentra distribuido por todos los desarrolladores del proyecto.

En el módulo estadísticas fue más compleja la generación de reportes aún con el uso del componente, pues aunque el mismo se encarga de llevar a cabo los procesos principales necesarios en el módulo, para su realización necesita la especificación detallada de datos que requieren un conocimiento mayor del funcionamiento de la librería JasperReports. Además, dicho módulo incluye funcionalidades que se pueden cumplir perfectamente explotando en gran medida las prestaciones de la librería y la herramienta iReport, por lo que se decidió excluir su implementación del componente y por lo tanto se hizo necesario su manejo por parte del programador.

Ejemplo de ello es la utilización de tablas cruzadas para el crecimiento de datos tanto vertical como horizontal y la utilización de variables a fin de dar cumplimiento a los requisitos estadísticos del módulo.

Sin embargo, la flexibilidad intrínseca a los reportes estadísticos fue manejada por el componente a través de la interpretación y procesamiento de los datos suministrados como los elementos a mostrar en el gráfico, el formato a darle al reporte, los datos a mostrar en los reportes según personalizaciones realizadas por el cliente, la forma de agrupar dichos datos, entre otras.

PRUEBAS REALIZADAS

Para su liberación, la aplicación desarrollada por el proyecto CICPC fue sometida a un gran número de pruebas entre las que se encuentran: Pruebas de Calidad Interna realizadas por el equipo de calidad perteneciente al proyecto CICPC, pruebas cruzadas realizadas por el equipo de desarrollo de CICPC y pruebas piloto y de aceptación realizadas con los usuarios finales de la aplicación.

Las pruebas enfocadas en la medición del cumplimiento de las funcionalidades identificadas fueron las realizadas por el equipo del proyecto y sus resultados se muestran a continuación:

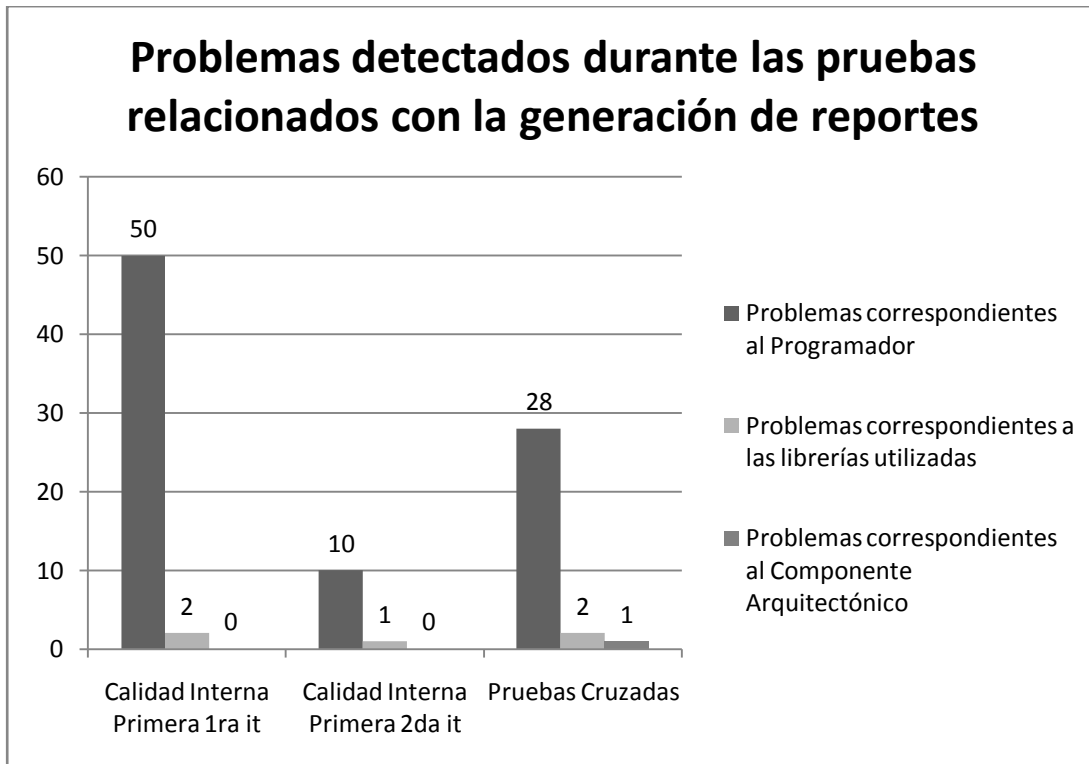


Figura 4.2: Problemas detectados durante las pruebas relacionados con la generación de reportes

Como se observa en el gráfico anterior, los principales problemas detectados en cuanto a la generación de reportes en la aplicación de CICPC, se derivan de un mal uso del componente por parte del programador. Estos problemas se resumen principalmente en descuidos en el momento de diseñar el reporte, escribiendo datos con errores ortográficos, o dando un mal diseño a los elementos del mismo.

Aunque se salen del ámbito del presente trabajo, el análisis de estos errores demuestra que a pesar de haber simplificado en alto grado el trabajo de los programadores en cuanto a generación de reportes, aún la realización de este proceso presenta fallas que deben solventarse al contribuir a un mayor conocimiento del tema por parte del equipo del proyecto.

En cuanto a los problemas detectados correspondientes a las librerías utilizadas en la implementación de la solución, demuestran que aún falta camino por recorrer en cuanto a las herramientas generadoras de reportes en java, teniendo en cuenta que la selección realizada tuvo su fundamento en el auge de las mismas y su uso por una amplia comunidad de usuarios así como en el cumplimiento de

las funcionalidades necesarias y, sin embargo, los problemas que presentan se hicieron evidentes durante las pruebas realizadas.

El problema detectado al componente estuvo dado por una mala definición en la plantilla de uno de los grupos de reportes identificados, el cual fue solventado con modificaciones precisas en su diseño.

Realizando una comparación con los problemas detectados a la aplicación en general se obtienen los siguientes resultados:

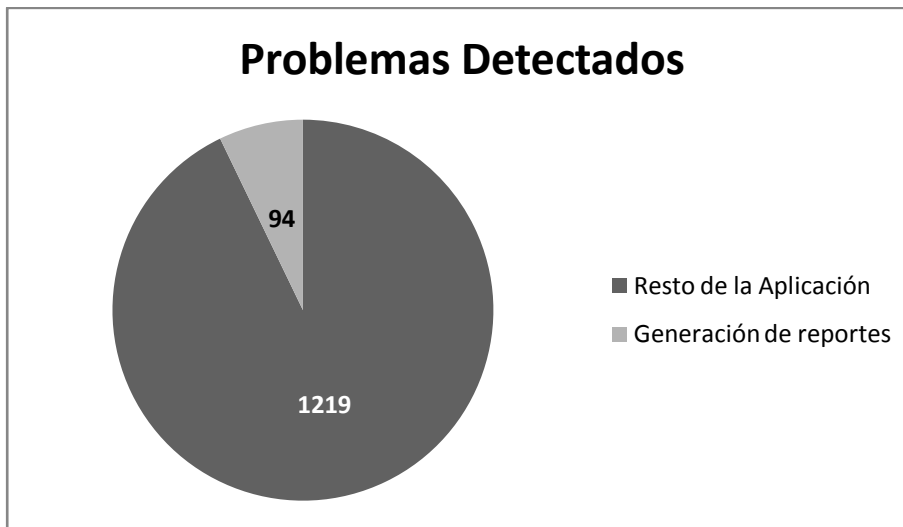


Figura 4.3: Problemas detectados a la aplicación

Demostrándose que aunque la generación de reportes está presente en la mayoría de los procesos gestionados por la aplicación, los errores detectados en la misma representan una parte pequeña del total.

Las pruebas enfocadas en el cumplimiento de las necesidades del usuario final fueron realizadas por el mismo arrojando los siguientes resultados:

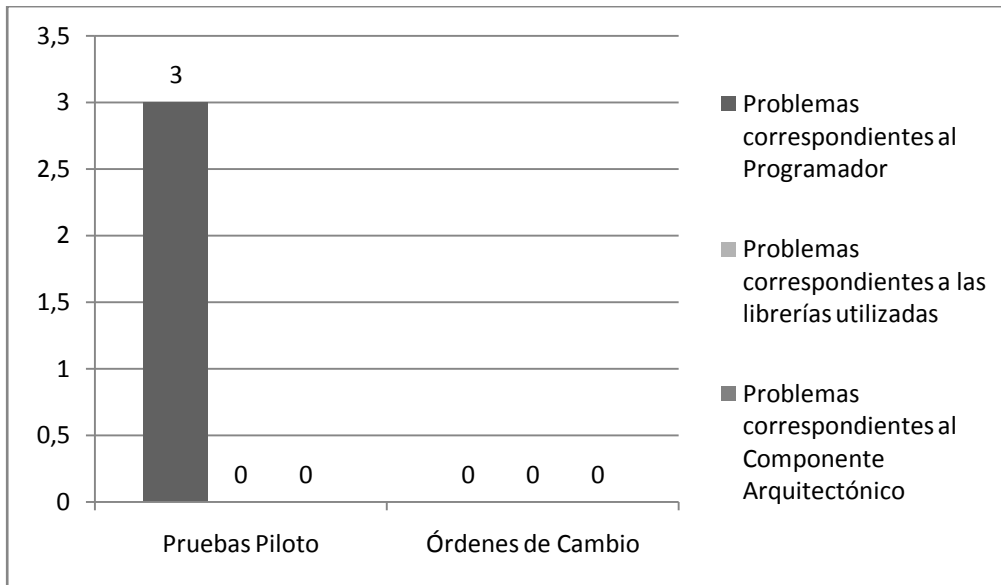


Figura 4.4: Problemas detectados durante las pruebas relacionados con la generación de reportes por el usuario final

El análisis del gráfico anterior demuestra, en primer lugar, que los errores que se fueron detectando a través de las etapas de pruebas por las que pasó la aplicación del CICPC fueron solventándose gradualmente hasta alcanzar un producto prácticamente libre de errores en lo que a reportes se refiere.

Para realizar mediciones de tiempo se utilizó JMeter, herramienta de carga para llevar a cabo simulaciones sobre cualquier recurso de Software. La misma fue inicialmente diseñada para pruebas de estrés en aplicaciones web, aunque su arquitectura ha evolucionado no sólo para llevar a cabo pruebas en componentes habilitados en Internet (HTTP), sino además en Bases de Datos , programas en Perl , requisiciones FTP y prácticamente cualquier otro medio. Posee la capacidad de realizar desde una solicitud sencilla hasta secuencias de requisiciones que permiten diagnosticar el comportamiento de una aplicación en condiciones de producción. En este sentido, simula todas las funcionalidades de un Navegador ("Browser"), o de cualquier otro cliente, siendo capaz de manipular resultados en determinada requisición y reutilizarlos para ser empleados en una nueva secuencia. (24) Los resultados obtenidos se resumen en la siguiente tabla:

Muestras	Media	Mediana	Línea del 90%	Min	Max	kb/sec
500	1001 ms	1005 ms	1008 ms	911 ms	1115 ms	58,7

Se debe destacar entonces el tiempo que demoran los reportes en ser generados, el cual oscila por el valor de 1 segundo, concluyendo que los reportes son presentados al usuario en un tiempo aceptable.

Además, los clientes estuvieron complacidos con la generación de sus informes que se convertía en un proceso general que daba como resultado reportes homogéneos en un corto lapso de tiempo, con un formato en correspondencia a las características de la institución y la formalidad y calidad requeridas.

4.4. ASPECTOS FAVORABLES

UTILIZACIÓN DE SOLUCIONES RECIENTES E INNOVADORAS

Para la creación del componente se utilizaron en la medida de lo posible soluciones recientes e innovadoras y con una amplia comunidad de usuarios, integrando tecnologías en lugar de realizar un desarrollo propio. Esto se manifiesta mediante estadísticas relacionadas con las descargas de las distintas librerías utilizadas.

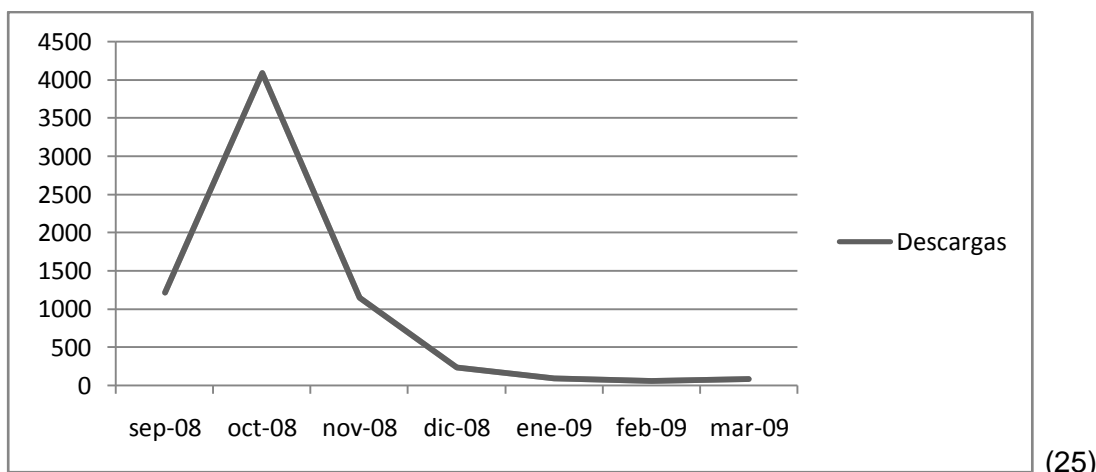


Figura 4.5: Descargas de jasperreports-3.1.0 en el último período

Como se observa en el gráfico anterior, en el período desde septiembre-08 hasta marzo-09 las descargas realizadas de la librería JasperReports en su versión 3.1.0 sobrepasan incluso las 4000 en uno de los meses. El primer ascenso de la gráfica se debe a la aparición de dicha versión y su descenso viene dado por el surgimiento de una versión superior.

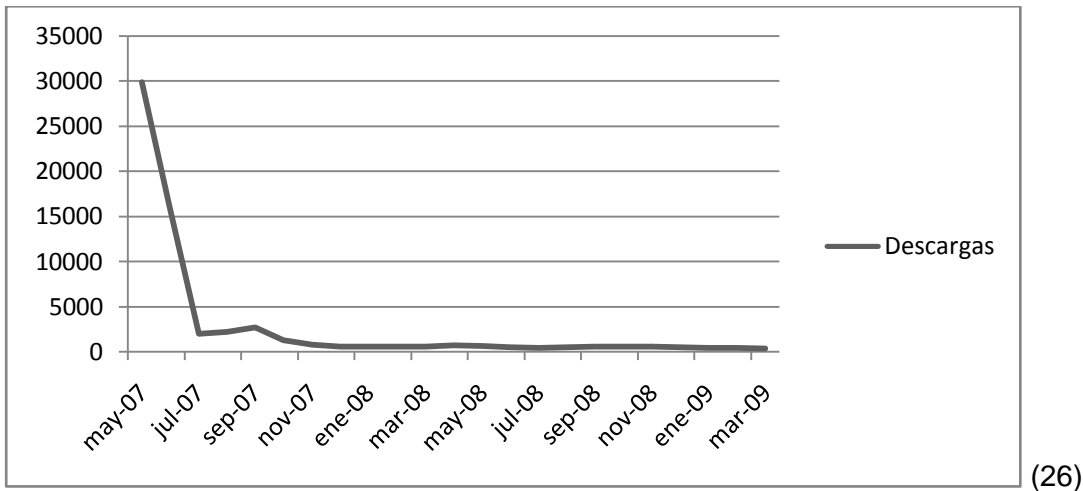


Figura 4.6: Descargas de iReport-1.3.3 en el último período

Como se observa en el gráfico anterior, en el período desde mayo-07 hasta noviembre-07 las descargas realizadas de la herramienta iReport en su versión 1.3.3 alcanzan incluso las 30000 en uno de los meses. Actualmente las descargas son mínimas debido al surgimiento de un gran número de nuevas versiones. Sin embargo, es preferible mantener la versión con que se cuenta debido a que las nuevas incluyen características y formas de diseñar los reportes diferentes; lo cual puede afectar los diseños que ya se encuentran incorporados al proyecto.

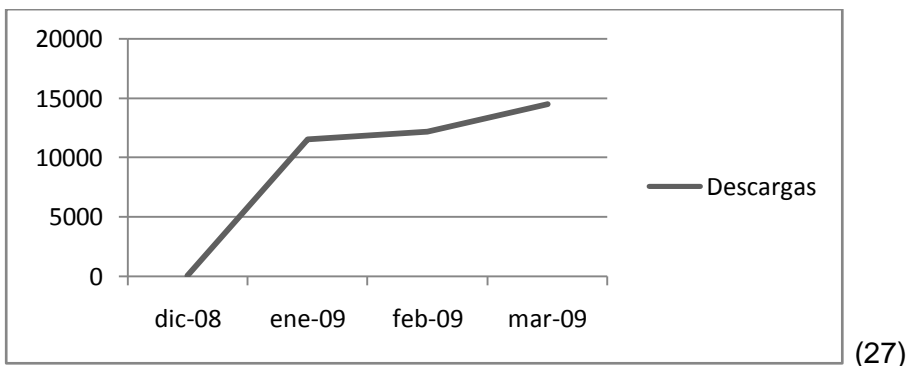


Figura 4.7: Descargas de jfreechart-1.0.12 en el último período

Como se observa en el gráfico anterior en el período desde diciembre-08 hasta marzo-09 las descargas realizadas de la librería jfreechart en su versión 1.0.12 sobrepasan incluso las 14000 en uno de los meses. El ascenso de la gráfica se debe a la aparición de dicha versión que constituye la más actual hasta la fecha.

Los datos antes mencionados validan por sí solos las herramientas y librerías utilizadas y aseguran la preferencia de una amplia comunidad de desarrolladores por las mismas.

SOLUCIONES A PROBLEMAS DETECTADOS EN LAS LIBRERÍAS

Entre las librerías antes mencionadas se encuentra JasperReports, a la cual se le detectaron una serie de problemas durante el proceso de desarrollo. En primer lugar cuando se agrega un estilo a un elemento del reporte, éste no cambia ni el backcolor ni el forecolor asociado al estilo, por lo cual se hizo necesario la sustitución de los atributos backcolor y forecolor del elemento por los provenientes del estilo pasado como parámetro.

Además no existe una banda que se imprima debajo del pageHeader y que se muestre solamente en la primera página, lo cual impide mostrar datos específicos asociados a la lista de entidades a listar al encabezar la misma solamente en la primera página. Para dar solución a dicho problema se utilizan los grupos definidos en JasperReports, se crea un primer grupo para colocar los datos que se desee mostrar solamente al comienzo del reporte pero debajo de la cabecera de la página y un segundo grupo para contener las cabeceras de columna de la lista a mostrar, especificándole a este último que se imprima en todas las páginas.

Otra de las deficiencias detectadas está dada por el hecho de que cuando los datos contenidos en el diseño del reporte exceden el tamaño de página definido, JasperReports lanza un error y no prevé solución para dicho problema. Para la solución también se utilizan los grupos, creando tantos como sea necesario para distribuir entre todos la información del detail y que la misma se imprima en su página correspondiente.

También, cuando un subreporte tiene como posición relativa flotante, no muestra los datos si estos exceden el espacio vertical asignado en el reporte maestro. Sin embargo, esta posición relativa es obligatoria para aquellos que se encuentran debajo de elementos que se pueden estirar con el fin de evitar que sus datos se superpongan. Para dar solución a este problema también se utilizaron los

grupos, teniendo en cuenta que un grupo no comienza a imprimirse hasta que no termine la banda que le precede, el subreporte puede ser colocado en un grupo con una posición fija y aún así se desplaza en consecuencia de los datos mostrados con anterioridad.

Se detectó además que la librería no procesa todos los caracteres HTML existentes, por lo cual se hizo necesaria la eliminación en las cadenas de los caracteres no reconocidos. Esto, aunque constituye una solución funcional, no responde a todos los requerimientos actuales debido a que muchos caracteres HTML no tienen su recíproco para JasperReports y simplemente son obviados en la impresión de los datos.

NUEVA DOCUMENTACIÓN DE LA LIBRERÍA

Además de la solución a los distintos problemas detectados en la librería, con el desarrollo de este trabajo quedan documentados algunos elementos que anteriormente no se podían encontrar. Esto posibilita que para muchos de los problemas que se presentan durante el desarrollo con las librerías seleccionadas, la solución se puede encontrar a partir del estudio de este documento.

La creación de gráficos de forma dinámica en tiempo de ejecución, aunque constituye una funcionalidad brindada por JasperReports, no cuenta con un soporte de documentación, sin embargo, en el presente trabajo se expone un estudio de la estructura de los mismos y la forma de crearlos que puede servir para cualquier proyecto que lo desee utilizar.

El cambio de los estilos de las letras en consecuencia con la codificación de los archivos PDF es otra de las funcionalidades de la librería que se encuentra sin documentar y de la que el presente trabajo realiza una breve exposición.

El uso de la librería richfaces en conjunto con JSF provoca la necesidad de modificar los métodos brindados por la clase JasperExportManager para mostrar un archivo adjunto por pantalla. La forma de modificación de dichos métodos también se encuentra documentada en el presente trabajo.

En toda la bibliografía relacionada con JasperReports se habla de la construcción de los diseños de reportes a partir de la creación de archivos *.jrxml*, ya sea de forma manual o haciendo uso de la herramienta para el diseño iReport. En el presente trabajo se muestra una forma mucho más dinámica de manipular los reportes y su diseño a partir de la clase JasperDesign. En el mismo se muestra una vista global de dicha clase y sus relaciones así como las posibilidades de uso de la misma. Se brindan

ejemplos funcionales de modificación de los diseños que proveen a los reportes de un alto nivel de flexibilidad dotando a los desarrolladores de una forma eficaz de cambiar las propiedades de los reportes en tiempo de ejecución de acuerdo con selecciones realizadas por el usuario.

Existe una librería que implementa una solución similar denominada DynamicJasper, cuya función es construir los reportes en tiempo de ejecución. Con la solución brindada por el presente trabajo se hace innecesario la inclusión de una librería intermedia teniendo en cuenta que todas las prestaciones que brinda se pueden obtener manipulando los objetos de la librería base JasperReports.

En otra dirección, el componente desarrollado es extensible y reutilizable en otros entornos de trabajo en java que utilicen el framework JSF para manejar la lógica de presentación. Es realmente sencillo y rápido implementar nuevas funcionalidades pues ya se tienen determinados los aspectos claves para la generación de reportes. Incluso, con la utilización de las funcionalidades ya implementadas se pueden construir componentes más generales que no dependan de diseños de reportes previos, sino que sean capaces de generar los propios.

4.5. ASPECTOS DESFAVORABLES

El principal aspecto negativo ha sido el tiempo de aprendizaje requerido para utilizar las tecnologías empleadas, lo cual ha causado grandes retrasos en la construcción del componente. Debido a la inexperiencia del equipo de desarrollo se obtuvieron soluciones iniciales que no contaban con la calidad requerida y que han ido evolucionando a través del proceso.

La inestabilidad de las librerías JasperReports y JFreeChart y de la herramienta iReport, así como la cantidad de versiones que aparecen a corto plazo de tiempo, causó que durante todo el proceso de desarrollo fuera necesario hacer paradas en la implementación para valorar la factibilidad de la utilización de una nueva versión.

Además, la inexistencia en la documentación de las librerías de aspectos específicos de su utilización, provocó la necesidad de realizar grandes investigaciones en aspectos que ya se encontraban implementados.

Por otra parte, el hecho de JasperReports no reconocer más que tags HTML básicos presentes en las cadenas recibidas, lo cual provocó la necesidad de parsearlas de la forma explicada anteriormente; conllevó a obviar un elevado número de tags con información relevante para el diseño de la cadena.

4.6. CONCLUSIONES

En el presente capítulo se analizaron los resultados de la puesta en práctica del componente arquitectónico en el entorno de trabajo del proyecto CICPC, así como los aspectos favorables y desfavorables detectados durante su desarrollo y el cumplimiento de los objetivos propuestos al inicio del trabajo. En el mismo se hace un análisis de los resultados obtenidos a partir de la realización de pruebas a la aplicación del CICPC y los señalamientos surgidos hacia los procesos de generación de reportes en las distintas partes del software.

CONCLUSIONES

El desarrollo del presente trabajo estuvo basado en un conjunto de etapas que fueron realizadas para dar cumplimiento a los objetivos específicos del mismo en pos de alcanzar las metas propuestas para su realización.

Se demostró la factibilidad de la utilización de la librería JasperReports al facilitar el cumplimiento de todas las funciones necesarias en el componente y permitir la obtención de reportes de un alto nivel con la estructura formal requerida por la organización que los va a utilizar.

La definición de las funcionalidades referentes a la generación de reportes que debía brindar el software a construir dotó al equipo de desarrollo de una visión de las necesidades actuales del proyecto.

El diseño del componente propuesto propició la estructuración del mismo a fin de lograr una delimitación de responsabilidades en sus clases, dotándolo de gran extensibilidad y garantizando su reutilización.

Con la implementación de la estructura necesaria para el componente propuesto, se hizo ineludible la búsqueda de alternativas a las funcionalidades brindadas por la librería JasperReports a fin de generar reportes que cumplieran con los requisitos definidos. Dichas alternativas constituyen un aporte al seguimiento del ciclo de vida de los reportes que lo dotan de mayor flexibilidad y a la vez permite la toma de decisiones en cuanto al diseño de los reportes en tiempo de ejecución.

Finalmente, al comprobar la validez de la solución final, se evidenció el cumplimiento de los objetivos propuestos, dotando al proyecto CICPC de un componente arquitectónico para la generación de reportes que aceleró su proceso de desarrollo y posibilitó la construcción de un producto menos expuesto a errores.

Sin embargo, los logros del presente trabajo no se limitaron a la construcción del componente sino que se realizaron aportes tecnológicos y sociales en cuanto a la forma de uso de las herramientas generadoras de reportes seleccionadas que servirán de base para estudios posteriores.

RECOMENDACIONES

Con la culminación del presente trabajo, se dio cumplimiento a los objetivos propuestos, dotando al proyecto CICPC de un componente arquitectónico para la generación de reportes que cumple con todas las funcionalidades identificadas. Sin embargo, se realizan una serie de recomendaciones para enfocar el trabajo futuro en el mejoramiento de la propuesta realizada.

- Hacer extensible el uso del componente independientemente del framework utilizado en la capa de presentación con el fin de poder usarse en cualquier aplicación java que se desee construir.
- Buscar la vinculación del mismo con soluciones similares a fin de lograr un resultado final más robusto compuesto por la unión de los aspectos positivos de varias implementaciones.
- Generalizar el conocimiento vinculado a la generación de reportes teniendo en cuenta que es un tema poco estudiado en la universidad.

Por otra parte, el componente propuesto realiza todas sus funciones partiendo de un diseño previo dando la posibilidad de modificar solamente algunos elementos del mismo. Por esta razón, se recomienda la construcción de una herramienta que permita la creación visual y en tiempo de ejecución del diseño de los reportes; para lo cual pueden usarse las funcionalidades implementadas en el componente que se propone agregando nuevas en la estructura del mismo.

Dicha herramienta debería cumplir básicamente con las siguientes prestaciones:

- Posible de insertar en cualquier aplicación Web.
- Una interfaz visual de fácil interacción para el usuario que le permita construir el diseño de los reportes de forma interactiva, teniendo en cuenta que en este caso el usuario puede ser una persona sin conocimientos de informática y por lo tanto se le debe facilitar la forma de definir los datos a mostrar en el reporte construyendo un modelo entendible por el mismo a través del modelo de datos existente en la aplicación.
- El diseño formará un objeto JasperDesign para continuar el ciclo de vida de los reportes a partir del mismo.
- El objeto podrá archivarse y usarse posteriormente en cualquier momento.

La construcción de esta herramienta permitirá que los usuarios finales de las aplicaciones desarrolladas que la incluyan sean capaces de generar sus propios reportes en cualquier momento sin tener que depender de estructuras previamente definidas y a la vez contar con un potente gestor de reportes encargado de velar por la calidad de los mismos y el cumplimiento de todo su ciclo de vida.

REFERENCIAS BIBLIOGRÁFICAS

1. **Frómata, Maykell.** *Arquitectura Candidata CICPC*. UCI : albet, 2008.
2. Sun Developer Network. [En línea] Sun Microsystems. [Citado el: 13 de marzo de 2009.] <http://developers.sun.com/>.
3. **Darwin, Ian F. y Brittain, Jason.** *Tomcat 6.0. La Guía Definitiva*. 2008. ISBN: 8441524319.
4. Starsur's Weblog. *Subversion*. [En línea] 12 de julio de 2008. [Citado el: 13 de marzo de 2009.] <http://starsur.wordpress.com/category/subversion/>.
5. WareSeeker. *Visual Paradigm for UML*. [En línea] [Citado el: 13 de marzo de 2009.] <http://wareseeker.com/Home-Education/Visual-Paradigm-for-UML-CE-%5BWindows%5D-6.3.zip/60217886>.
6. La flecha: tu diario de ciencia y tecnología. *Red Hat presenta "Developer Studio" basado en Eclipse*. [En línea] 27 de agosto de 2007. [Citado el: 13 de marzo de 2009.] <http://www.laflecha.net/canales/softlibre/red-hat-presenta-developer-studio-basado-en-eclipse/>.
7. Open Source Software Engineering Tools. *Subclipse*. [En línea] [Citado el: 13 de marzo de 2009.] <http://subclipse.tigris.org/>.
8. **González, Héctor Suárez.** *Manual Hibernate*. [En línea] 2003. [Citado el: 13 de marzo de 2009.] <http://www.javahispano.org/contenidos/archivo/77/ManualHibernate.pdf>.
9. This Spring...Get ready for Spring! [En línea] [Citado el: 13 de marzo de 2009.] <http://www.theserverlabs.com/folletos/Folleto%20Spring.pdf>.
10. **Juan Medín Piñeiro, Antonio García Figueras.** *Hacia una arquitectura con JavaServer Faces, Spring, Hibernate y otros frameworks*. [En línea] junio de 2006. [Citado el: 13 de marzo de 2009.] http://www.csi.map.es/csi/tecniap/tecniap_2006/01T_PDF/hacia%20una%20arquitectura.pdf.
11. JavaHispano. *Versión 0.7 de Acegi Security System for Spring*. [En línea] 24 de enero de 2005. [Citado el: 13 de marzo de 2009.]

http://javahispano.org/contenidos/es/versrion_0_7_de_acegi_security_system_for_spring/?menuld=tag&onlypath=true.

12. **Berzal, Fernando**. Pruebas de Unidad con JUnit. *Técnicas útiles en el desarrollo del software*. [En línea] [Citado el: 13 de marzo de 2009.] <http://elvex.ugr.es/decsai/java/pdf/8C-JUnit.pdf>.

13. SAP. *Crystal Reports para Eclipse*. [En línea] [Citado el: 13 de marzo de 2009.] <http://www.sap.com/mexico/solutions/sapbusinessobjects/sme/reporting/eclipse/index.epx>.

14. Eclipse. *BIRT Project*. [En línea] Eclipse Foundation. [Citado el: 13 de marzo de 2009.] <http://www.eclipse.org/birt/phoenix/>.

15. **Heffel, David R**. *JasperReports for Java Developers*. Birmingham : Packt Publishing Ltd., 2006. ISBN 1-904811-90-6.

16. **Sadik, Hossam**. FCI-H: The developers ultimate resource. *Java Reporting 2*. [En línea] 6 de septiembre de 2008. [Citado el: 25 de marzo de 2009.]

17. www.jfree.org. [En línea] [Citado el: 29 de octubre de 2008.] www.jfree.org/jfreechart.

18. SourceForge. *A Design Tool iReport for JasperReports*. [En línea] [Citado el: 13 de marzo de 2009.] <http://ireport.sourceforge.net/manual0.2.0.html>.

19. **Amaro Calderón, Sarah Dámaris, Valverde Rebaza, Jorge Carlos**. *Metodologías Ágiles*. Trujillo-Perú : Universidad Nacional de Trujillo, Facultad de Ciencias Físicas y Matemáticas, Escuela de Informática, 2007.

20. Programación Extrema . *Objetivos de la programación extrema*. [En línea] <http://extre.blogspot.com/2006/11/objetivos-de-la-programacin-extrema.html>.

21. **Beck, Kent**. *Extreme Programming Explained*. s.l. : Addison Wesley, 1999. ISBN: 0201616416.

22. **Beck, Kent y Fowler, Martin**. *Planning Extreme Programming*. s.l. : Addison Wesley, 2000. ISBN: 0-201-71091-9.

23. **González, Carlos Sánchez**. *ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras*. s.l. : UNIVERSIDADE DA CORUÑA, 2004.

24. Osmosis Latina. *JMeter*. [En línea] 20 de octubre de 2005. <http://www.osmosislatina.com/jmeter/basico.htm>.
25. SourceForge.net. *Project Statistic for JasperReports - Java Reporting*. [En línea] 2009. http://sourceforge.net/project/stats/detail.php?group_id=36382&ugn=jasperreports&type=prdownload&mode=week&package_id=28579&release_id=626893&file_id=1628202.
26. SourceForge.net. *Project Statistics for iReport - Designer for JasperReports*. [En línea] 2009. http://sourceforge.net/project/stats/detail.php?type=prdownload&group_id=64348&ugn=ireport&package_id=263428&release_id=680503&file_id=1628565.
27. SourceForge.net. *Project Statistics for JFreeChart*. [En línea] 2009. http://sourceforge.net/project/stats/detail.php?type=prdownload&group_id=15494&ugn=jfreechart&package_id=12428&release_id=677082&file_id=1619239.

BIBLIOGRAFÍA

Albiol, F. R. (2004). *JasperReports, iReport y Subreportes*.

Amaro Calderón, S. D. (2007). *Metodologías Ágiles*. Trujillo-Perú: Universidad Nacional de Trujillo, Facultad de Ciencias Físicas y Matemáticas, Escuela de Informática.

Baird, S. (2003). *Sams Teach Yourself Extreme Programming in 24 Hours*. Sams Publishing.

Basurto, C. K. (s.f.). *Adictos al Trabajo*. Recuperado el 2007, de Introducción a iReport.

Beck, K. (1999). *Extreme Programming Explained*.

Beck, K., & Fowler, M. (2000). *Planning Extreme Programming*. Addison Wesley .

Berzal, F. (s.f.). *Pruebas de Unidad con JUnit*. Recuperado el 13 de marzo de 2009, de Técnicas útiles en el desarrollo del software: <http://elvex.ugr.es/decsai/java/pdf/8C-JUnit.pdf>

Business Objects. *Crystal Reports for Eclipse: Getting Started Guide*.

Crick, J. (2004). *Crystal Reports Server XI Visión general de su funcionalidad*. Business Object.

Danciu, T. (2002). *The JasperReports Ultimate Guide*.

Duharte, F. R., Correa, J. C., & Zaila, Y. B. (2006). *Reportes con JasperReports, iReport y JFreeChart en aplicaciones empresariales en java*. Ciudad de la Habana: UCI.

Escribano, G. F. (2002). *Introducción a Extreme Programming*.

Gilbert, D. (2004). *The JFreeChart Class Library*.

González, C. S. (2004). *ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras*. UNIVERSIDADE DA CORUÑA.

Heffel, D. R. (2006). *JasperReports for Java Developers*. BIRMINGHAM - MUMBAI: packt.

Hibernate. (s.f.). Recuperado el 2007, de iReport with hibernate support preview.

Jabato: Apuntes de contenido técnico. (s.f.). Recuperado el 2007, de Generación de informes.

JasperAssistant User Guide. (2005). Infologic.

Jeffries, R., Anderson, A., & Hendrickson, C. (2000). *Extreme Programming Installed.* Addison Wesley .

Juan Medín Piñeiro, A. G. (junio de 2006). *Hacia una arquitectura con JavaServer Faces, Spring, Hibernate y otros frameworks.* Recuperado el 13 de marzo de 2009, de http://www.csi.map.es/csi/tecniap/tecniap_2006/01T_PDF/hacia%20una%20arquitectura.pdf

Khramtchenko, S. (2004). *Comparing eXtreme Programming and Feature Driven Development in academic and regulated environments.* Harvard University: Software Architecture and Engineering.

Marchesi, M. *The New XP.*

Mora, R. C. (2006). *Adictos al Trabajo.* Recuperado el 2007, de Gráficas en Java con JFreeChart.

Programación Extrema . (s.f.). Recuperado el 2009, de Objetivos de la programación extrema: <http://extre.blogspot.com/2006/11/objetivos-de-la-programacin-extrema.html>

Sadik, H. (6 de septiembre de 2008). *FCI-H: The developers ultimate resource.* Recuperado el 25 de marzo de 2009, de Java Reporting 2.

Sánchez, E. A., Letelier, P., & Canós, J. H. *Mejorando la gestión de historias de usuario en eXtreme Programming.* Valencia - España.

Wells, D. (17 de febrero de 2006). *Extreme Programming: A gentle introduction.* . Recuperado el enero de 2009, de <http://www.extremeprogramming.org/index.html>

GLOSARIO DE TÉRMINOS

Apache Software Foundation: Organización no lucrativa creada para dar soporte a los proyectos de software bajo la denominación *Apache*, incluyendo el popular servidor HTTP Apache.

API: Una interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Atomicidad: Asegura que un conjunto de cambios ingresa completamente al repositorio o no ingresa. No se realizan cambios parciales causados por errores de conexión.

BackColor: Obtiene o establece el color de fondo.

Classpath: Determina dónde buscar tanto las clases o librerías de Java (el API de Java) como otras clases de usuario.

Diff binarios: Las diferencias entre archivos se tratan de igual forma así sean archivos de texto o archivos binarios.

DTD: Descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD.

ForeColor: Obtiene o establece el color de primer plano.

Introspection: Capacidad de algunos lenguajes de programación orientada o objetos para determinar el tipo del objeto en tiempo de ejecución.

JavaServer Pages: Tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

JDBC: API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

Parsear HTML: Limpiar el código HTML sacando código basura o redundante, o modificando la sintaxis de código mal utilizado.

Plain Old Java Objects: Clases simples y que no dependen de un framework en especial.

Programación Orientada a Aspectos: Paradigma de programación cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos.

Proyecto Jakarta: Encargado de crear y mantener todas las soluciones open Source creadas para la plataforma Java. Los productos del proyecto se dividen en tres categorías generales: bibliotecas, herramientas y APIs, motores y aplicaciones del lado del Servidor.

Richfaces: Librería de componentes para JSF para facilitar la integración a las capacidades de AJAX dentro del desarrollo de aplicaciones.

Servlets: Objeto que se ejecuta en un servidor o contenedor JEE, especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente HTML.

SGML: "Lenguaje de Marcado Generalizado". Consiste en un sistema para la organización y etiquetado de documentos.

SOA: Arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio.

Software de middleware: Software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

Spike: Programa simple que explora soluciones potenciales.

Unified Expression Language: Representa la unión del lenguaje ofrecido por JSP 2.0 y el creado por la tecnología JavaServer Faces.

XML: Metalenguaje extensible de etiquetas. Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

ANEXOS

ANEXO 1: Plantillas para Reportes de CICPC

República Bolivariana de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y Justicia.

COORDINACIÓN NACIONAL DE CIENCIAS FORENSES
DIVISIÓN DE ANTROPOLOGÍA FORENSE

FICHA ANTROPOLÓGICA PARA CADÁVERES NO IDENTIFICADOS

NRO. DE CADÁVER:	FECHA:
PROCEDENCIA:	ANTROPOLOGO:

VARIABLES ANTROPOLÓGICAS GENERALES

SEXO:	TIPOLOGÍA RACIAL:
EDAD:	BIOTIPO CONSTITUCIONAL:
ESTATURA:	CALOR DE PIEL:

DESCRIPCIÓN SOMATOSCÓPICA

CONFIGURACIÓN DEL ROSTRO:

CABELLOS:

FRENTE:

ARCOS SUPERCILIARES:

REGIÓN GLABELAR:

PESTAÑAS:

CEJAS:

LABIOS:

BOCA:

ESPAZIO ENTRE BORDE DEL LABIO INFERIOR Y REBORDE MENTONIANO:

MENTÓN:

POMULOS:

OREJAS:

SISTEMA PILOSO:

BIGOTES:

MAMAS (SENOS):

MANOS:

PIES:

Gobierno Bolivariano de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y Justicia | Consejo de Investigaciones Científicas, Técnicas y Tecnológicas

República Bolivariana de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y Justicia.

REPORTE DE SISTEMA

(Nombre del dte.) (Dte) de (Mes) de (Año)
(Tipo de Despacho) (Nombre de Despacho)
(Estado Geográfico)

IDENTIFICACIÓN PERSONAL

CRIMINOSO

NOMBRES:	Julian
APELLIDOS:	Yanez
CECULA:	124887

RESUMEN

NOMBRES	APELLIDOS	CECULA	FECHA NACIMIENTO
Julian	Roman	Y - 1248878	12 / 04 / 89
Julian	Pedro	Y - 1248248	12 / 04 / 78
Julian	Roman	Y - 1248248	12 / 04 / 87

(Nombre y Apellido del Funcionario)
(Cargo) (Orden de) (Por Orden)

Gobierno Bolivariano de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y Justicia | Consejo de Investigaciones Científicas, Técnicas y Tecnológicas

Pág (Total) de (Total)

República Bolivariana de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y Justicia.

MEMORANDUM

(Nombre del dte.) (Dte) de (Mes) de (Año)
(Cargo) (Nombre de Despacho) (Estado Geográfico del Despacho)
A: (Nombre de Despacho) (Tipo de Despacho)
En Respuesta: (Por Comunicación) a la de (Fecha)

PARA: (Nombre y Apellido del Funcionario destino)
(Cargo) (Nombre de Despacho) (Estado Geográfico del Despacho)

DE: (Nombre y Apellido del Funcionario destino)
(Cargo) (Nombre de Despacho) (Estado Geográfico del Despacho)

ASUNTO: (Asunto de la comunicación)

(Resumen)

NOMBRES Y APELLIDOS: Alejandro Manuel Navarro Garcia
NACIONALIDAD: Venezolano
NO. DE CECULA: 1248878
DELITO: Robo
FECHA DE NACIMIENTO: 30 de noviembre de 1980
EDAD: 27 años

De conformidad a lo establecido en la ley: (Ley) (Artículo 1), (Artículo 2), ...

(Decreto)

(Nombre y Apellido del Funcionario destino)
(Cargo) (Nombre de Despacho) (Estado Geográfico del Despacho)

(Iniciales del creador) (Iniciales del revisor) (Iniciales del aprobador)

Gobierno Bolivariano de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y Justicia | Consejo de Investigaciones Científicas, Técnicas y Tecnológicas

Pág (Total) de (Total)

REPÚBLICA BOLIVARIANA DE VENEZUELA
CUERPO DE INVESTIGACIONES CIENTÍFICAS, PENALES Y CRIMINALÍSTICAS

(Tipo de Despacho) (Nombre de Despacho)

OFICIO

(Nombre del dte.) (Dte) de (Mes) de (Año)
(Cargo) (Nombre de Despacho) (Estado Geográfico del Despacho)

(Nombre y Apellido del Funcionario destino)
(Cargo) (Nombre de Despacho) (Estado Geográfico del Despacho)

Asunto:

Tengo a bien dirigirme a Usted para requerir su valerosa colaboración en el dictamen de un Orden de aprehensión sobre el ciudadano cuyo dato le acompaño.

NOMBRES Y APELLIDOS: Alejandro Manuel Navarro Garcia
NACIONALIDAD: Venezolano
NO. DE CECULA: 1248878
DELITO: Robo
FECHA DE NACIMIENTO: 30 de noviembre de 1980
EDAD: 27 años

De conformidad a lo establecido en la ley: (Ley) (Artículo 1), (Artículo 2), ...


(Decreto)

(Nombre y Apellido del Funcionario destino)
(Cargo) (Nombre de Despacho) (Estado Geográfico del Despacho)

(Iniciales del creador) (Iniciales del revisor) (Iniciales del aprobador)


Gobierno Bolivariano de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y Justicia | Consejo de Investigaciones Científicas, Técnicas y Tecnológicas

Pág (Total) de (Total)



REPÚBLICA BOLIVARIANA DE VENEZUELA
 GOBIERNO DE INVESTIGACIONES CIENTÍFICAS, PENALES Y CRIMINALÍSTICAS

[TIPO DE DE SPACHO] - [NOMBRE DE DE SPACHO]



INFORME DE [Tipo de Informe]
 [Nº Informe] [Nombre del caso] [Día] de [Mes] de [Año]
 [Acta Procesal] / [Nº Acta Procesal]
 En Respuesta: [Nº Comunicación a la que responde]

[Nombre de la Sección]
 [Descripción]

[Nombre de la Sección]
 [Descripción]


[Nombre de la Sección]
 [Descripción]

[Rango] [Nombres y Apellidos del Funcionario responsable]
 [Nombre del Despacho Emisor] [Estado Geográfico de Despacho]

[Rango] [Nombres y Apellidos del Funcionario responsable]	[Rango] [Nombres y Apellidos del Funcionario responsable]
---	---


[Nombre del Despacho Emisor] [Estado Geográfico del Despacho] [Nombre del Despacho Emisor] [Estado Geográfico del Despacho]

[Iniciales del creador] [Iniciales del revisor] [Iniciales del aprobador]
 [Fecha] [Hora]



Gobierno Bolivariano de Venezuela | Ministerio del Poder Popular para Relaciones Interiores y de Justicia | Centro de Investigaciones Científicas, Penales y Criminalísticas

Pág. [Total] de [Total]



ANEXO 2: Tareas de Implementación. Primera Iteración

Tarea	
Número tarea: 1	Número historia: 1
Nombre tarea: Encontrar recursos para el reporte	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 12 enero 2009	Fecha fin: 12 enero 2009
Descripción: Los recursos para generar los reportes se encuentran ubicados en la dirección WebContent/resources del proyecto. Deben cargarse desde allí en tiempo de ejecución haciendo uso del Contexto de JSF.	

Tarea	
Número tarea: 2	Número historia: 1
Nombre tarea: Insertar subreportes	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 13 enero 2009	Fecha fin: 13 enero 2009
Descripción: Los reportes pueden contener subreportes. Deben identificarse los subreportes que contiene un reporte general, cargar los mismos e incrustárselos en su diseño.	

Tarea	
Número tarea: 3	Número historia: 1
Nombre tarea: Incluir origen de datos genérico	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 14 enero 2009	Fecha fin: 15 enero 2009
Descripción: El origen de datos válido para generar los reportes de CICPC debe estar enfocado a las funcionalidades de cada una de sus entidades. Debe crearse un origen de datos genérico para las mismas.	

Tarea

Número tarea: 4	Número historia: 1
Nombre tarea: Unificar Código	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 16 enero 2009	Fecha fin: 16 enero 2009
Descripción: Debe unificarse el código resultante de las tareas anteriores y generar el reporte final.	

Tarea	
Número tarea: 5	Número historia: 2
Nombre tarea: Diseñar formatos generales de reportes	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 19 enero 2009	Fecha fin: 20 enero 2009
Descripción: Diseñar un formato que contenga tanto el encabezado como el pie de página para todos los tipos de reportes especificados en la historia de usuario.	

Tarea	
Número tarea: 6	Número historia: 2
Nombre tarea: Mezclar los elementos de dos formatos	
Tipo de tarea : Desarrollo	Puntos estimados: 3
Fecha inicio: 21 enero 2009	Fecha fin: 23 enero 2009
Descripción: Mezclar los elementos de cada una de las bandas de los dos reportes así como sus campos, parámetros y variables.	

Tarea	
Número tarea: 7	Número historia: 4
Nombre tarea: Exportar reporte a formato PDF.	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 26 enero 2009	Fecha fin: 27 enero 2009
Descripción: A partir de un reporte previamente generado exportarlo a formato PDF y mostrarlo por pantalla en una nueva ventana.	

Tarea	
Número tarea: 8	Número historia: 5
Nombre tarea: Exportar reporte a formato XLS.	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 28 enero 2009	Fecha fin: 29 enero 2009
Descripción: A partir de un reporte previamente generado exportarlo a formato XLS y mostrarlo por pantalla en una nueva ventana.	

Tarea	
Número tarea: 9	Número historia: 7
Nombre tarea: Cargar imagen	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 30 enero 2009	Fecha fin: 30 enero 2009
Descripción: Las imágenes que contienen los reportes se encuentran ubicados en la dirección WebContent/resources/images del proyecto. Deben cargarse desde allí en tiempo de ejecución haciendo uso del Contexto de JSF y devolverlas como un objeto URL.	

ANEXO 3: Tareas de Implementación. Segunda Iteración

Tarea	
Número tarea: 10	Número historia: 3
Nombre tarea: Identificar los caracteres y secuencias que reconoce JasperReports	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 9 febrero 2009	Fecha fin: 9 febrero 2009
Descripción: Los textos con estilos que muestra JasperReports deben tener una estructura reconocida por el mismo. Se deben identificar los elementos que conforman dicha estructura.	

Tarea	
Número tarea: 11	Número historia: 3
Nombre tarea: Eliminar del texto los caracteres no reconocidos.	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 10 febrero 2009	Fecha fin: 10 febrero 2009
Descripción: Según los elementos identificados que conforman la estructura de JasperReports, se deben eliminar aquellos caracteres que no sean reconocidos por el mismo.	

Tarea	
Número tarea: 12	Número historia: 3
Nombre tarea: Dar formato soportado por el PDF.	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 11 febrero 2009	Fecha fin: 12 febrero 2009
Descripción: Los PDF tienen características propias en cuanto a la forma de codificar el texto con estilo, por lo cual se debe dar al texto dicha codificación.	

Tarea	
Número tarea: 13	Número historia: 3
Nombre tarea: Cambiar la escala de tamaños.	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 13 febrero 2009	Fecha fin: 13 febrero 2009
Descripción: JasperReports mide el tamaño de los textos de forma particular, debe ajustarse la escala de los mismos a dicha forma.	

Tarea	
Número tarea: 14	Número historia: 6
Nombre tarea: Estudiar estructura de los Gráficos en JasperReports	
Tipo de tarea : Desarrollo	Puntos estimados: 1

Fecha inicio: 16 febrero 2009	Fecha fin: 16 febrero 2009
Descripción: Se debe estudiar la estructura de los gráficos como objetos y la forma de construir o modificar los mismos.	

Tarea	
Número tarea: 15	Número historia: 6
Nombre tarea: Generar Gráficos de pastel	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 17 febrero 2009	Fecha fin: 18 febrero 2009
Descripción: Construir objetos que representen gráficos de tipo pastel y pastel 3D.	

Tarea	
Número tarea: 16	Número historia: 6
Nombre tarea: Generar Gráficos con series de datos.	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 19 febrero 2009	Fecha fin: 20 febrero 2009
Descripción: Construir objetos que representen gráficos de barras, barras 3D, líneas, área y pila.	

Tarea	
Número tarea: 17	Número historia: 8
Nombre tarea: Concatenar elementos JasperPrint.	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 23 febrero 2009	Fecha fin: 23 febrero 2009
Descripción: Se debe generar un reporte general resultante de la unión de varios elementos JasperPrint generados.	

Tarea	
Número tarea: 18	Número historia: 9

Nombre tarea: Cambiar propiedades a elementos de texto.	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 24 febrero 2009	Fecha fin: 24 febrero 2009
Descripción: Se deben cambiar color de fondo, color de letra y tamaño de letra a los elementos de texto del reporte.	

Tarea	
Número tarea: 19	Número historia: 10
Nombre tarea: Crear grupo.	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 25 febrero 2009	Fecha fin: 26 febrero 2009
Descripción: Crear un grupo dentro del reporte a partir de la construcción de una expresión para el mismo según la selección del usuario.	

Tarea	
Número tarea: 20	Número historia: 10
Nombre tarea: Crear cabecera del grupo	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 27 febrero 2009	Fecha fin: 27 febrero 2009
Descripción: Crear una cabecera para el grupo que muestre en un elemento de texto el valor de la expresión por la que se está agrupando.	

Tarea	
Número tarea: 21	Número historia: 10
Nombre tarea: Crear pie del grupo	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 2 marzo 2009	Fecha fin: 2 marzo 2009
Descripción: Crear un pie para el grupo que contenga la cantidad de elementos agrupados según su expresión.	

Tarea	
Número tarea: 22	Número historia: 11
Nombre tarea: Cambiar propiedades a elementos de una tabla.	
Tipo de tarea : Desarrollo	Puntos estimados: 2
Fecha inicio: 3 marzo 2009	Fecha fin: 4 marzo 2009
Descripción: Cambiar propiedades de color de fondo, color de texto y tamaño de texto a las celdas de la tabla en dependencia del lugar que ocupen para que cumplan con los formatos especificados.	

Tarea	
Número tarea: 23	Número historia: 12
Nombre tarea: Eliminar Bandas	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 5 marzo 2009	Fecha fin: 5 marzo 2009
Descripción: Eliminar bandas en dependencia del tipo de reporte que se desee mostrar.	

Tarea	
Número tarea: 24	Número historia: 12
Nombre tarea: Modificar atributos de posicionamiento.	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 6 marzo 2009	Fecha fin: 6 marzo 2009
Descripción: Se debe lograr que los elementos dentro del reporte se acomoden de forma correcta ante la eliminación o adición de elementos vecinos.	