UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

REVISIÓN DE CALIDAD A UN SISTEMA DE GESTIÓN DE VALORES BURSÁTILES

INFORME DE TRABAJO PROFESIONAL QUE PARA OBTENER EL TÍTULO DE

Ingeniera en Computación

PRESENTA:
Gloria Hurtado Díaz

DIRECTOR:
M.I. Jorge Valeriano Assem

Ciudad Universitaria, Septiembre del 2009

Con todo mi agradecimiento, ustedes saben quiénes son, porque han estado a mi lado siempre…

… y muy especialmente a Fernando, sin tí esto era solo un proyecto y hoy se convierte en un logro en mi vida.

Detrás de cada línea de llegada, hay una de partida.

Detrás de cada logro, hay otro desafío.

Si extrañas lo que hacías, vuelve a hacerlo.

Sigue aunque todos esperen que abandones.

No dejes que se oxide el hierro que hay en ti.

Contenido

1 Objetivo 3
2 Antecedentes del proyecto....4
3 Definición del problema....5
4 Desarrollo 7
4.1 Etapa inicial 7
4.1.1 Reunión de inicio del proyecto 8
4.1.2 Entrega/recepción de productos del sistema generados a la fecha 8
4.1.3 Levantamiento de inventario de los entregables obtenidos 9
4.2 Etapa de ejecución 9
4.2.1 Fase de concepción 9
4.2.2 Fase de elaboración....10
4.2.3 Fase de construcción 10
4.2.4 Fase de transición....19
5 Análisis y metodología empleada....20
5.1 Modelo de McCall 21
5.2 Métricas de código 25
5.2.1 Clasificación de las métricas....25
5.2.1.1 Métricas de tamaño....26
5.2.1.2 Métricas de complejidad....26
5.2.1.3 Métricas de programación orientada a objetos 27
6 Participación profesional 30
7 Resultados y aportaciones....32
7.1 Revisión a soporte documental....32
7.1.1 Elementos considerados para la revisión 33
7.1.2 Referencias a estándares y guías utilizadas en la construcción de la documentación 33
7.1.3 Hallazgos y recomendaciones 34
7.1.4 Resumen de incidencias....39
7.2 Métricas de código 40
7.2.1 Métricas de tamaño del sistema....40
7.2.2 Métricas de complejidad del sistema 42
7.2.3 Métricas de programación orientada a objetos....43

7.3 Análisis de código fuente 49
7.3.1 Incidencias factibles de adecuar a corto plazo 49
7.3.2 Incidencias factibles de adecuar a mediano plazo 52
7.3.3 Incidencias factibles de adecuar a largo plazo 55
7.3.4 Resumen total de incidencias 60
8 Conclusiones 63
9 Referencias 64

1 Objetivo

El objetivo general de este informe es dar a conocer las actividades realizadas en la revisión de calidad a un sistema de software ya construido, para mejorar sus factores de calidad con la finalidad de garantizar su óptima operación y mantenimiento. La importancia de este trabajo radica en la aplicación de los conocimientos de ingeniería de software para poder evaluar el sistema y posteriormente establecer las recomendaciones de cambio necesarias para mejorar su funcionamiento.

Los objetivos específicos que se definieron para llevar a cabo la revisión al sistema son:

- Realizar un proceso de aseguramiento de calidad a un producto de software, el cual incluya las siguientes actividades:
    - Revisar y analizar la documentación del sistema en las categorías de visión, ámbito, especificación y diseño
    - Revisar y analizar el código fuente del sistema para identificar su tamaño, complejidad, estructura, dependencia y las desviaciones o no conformidades de apego a estándares y convenciones de estilo
- Identificar los riesgos en los factores de calidad del sistema con base en las revisiones
- Generar las recomendaciones y sugerencias necesarias de acuerdo a los riesgos identificados
- Elaborar y presentar los informes técnicos y de resultados.

2 Antecedentes del proyecto

El cliente es una empresa privada que opera como un depósito central de valores en México, proporciona los servicios de custodia y administración de valores, con un alcance nacional e internacional de transferencia electrónica de valores y efectivo, así como servicios de información. Como resultado de una estrategia de modernización de su infraestructura tecnológica promovida por la dirección operativa, esta empresa decidió realizar la migración de sus sistemas que dan soporte a su operación, a una nueva plataforma en el periodo de principios del año 2006 a mediados del 2008, con planes de liberación de sus sistemas en el segundo semestre del 2008.

Debido a este hecho, la empresa requería contar con un nivel de certeza adecuado para que dichos sistemas, que fueron desarrollados y que serían implementados para reemplazar a los existentes, fueran los más precisos y estables posibles, con la finalidad de garantizar la continuidad en la operación y que las funcionalidades desarrolladas, principalmente las que manejan valores y efectivo, estuvieran lo más libres de defectos, para que las transacciones fueran seguras y confiables, debido a que el impacto ocasionado por alguna falla durante la operación podría significar pérdidas económicas considerables, así como problemas legales por un mal manejo de información, ya que estos sistemas se rigen por una base legal bien establecida.

3 Definición del problema

Para mitigar el riesgo que representaba para el cliente implementar el sistema en operación, se identificó un área de oportunidad para realizar las revisiones de calidad a su sistema, el cual pudieran mostrar de manera cuantitativa y verificable, el comportamiento actual, y encontrar todos los posibles defectos o comportamientos no esperados, para ajustarlos antes de su implantación.

Debido al gran tamaño del sistema y a que son varios módulos los que lo conforman, para los fines de este informe solo se consideraron los resultados de la revisión de uno de los módulos componentes, el cual es denominado el Módulo de Operación de Efectivo, conocido convencionalmente como MOS.

El Módulo de Operación de Efectivo (MOS) forma parte del Sistema de Depósito y Liquidación de Valores (SDLV), y procesa instrucciones de liquidación que involucran únicamente entrega de efectivo. El MOS aplica una serie de reglas y controles que preparan las instrucciones recibidas para su liquidación en cuentas de efectivo que se mantiene a favor de sus participantes.

En la Figura 1 se muestra al MOS como parte del Sistema de Depósito y Liquidación de Valores

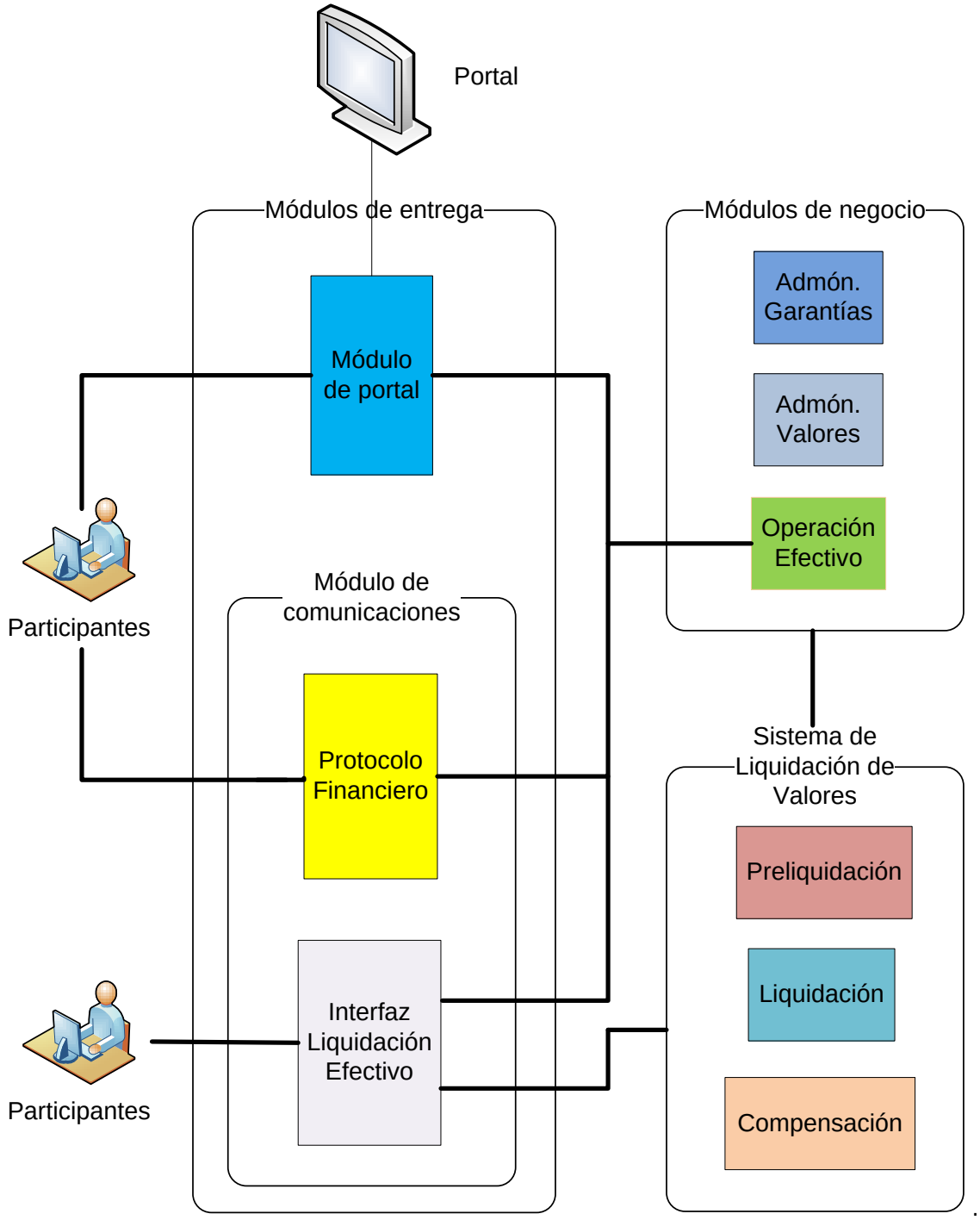


Figura 1 Arquitectura general del sistema.

4 Desarrollo

El desarrollo de los trabajos de revisión del MOS se llevó a cabo en tres etapas, las cuales fueron definidas con la finalidad de permitir organizar de una mejor manera las actividades a realizar. En la Figura 2 se muestran de manera general las actividades a realizar en cada etapa.

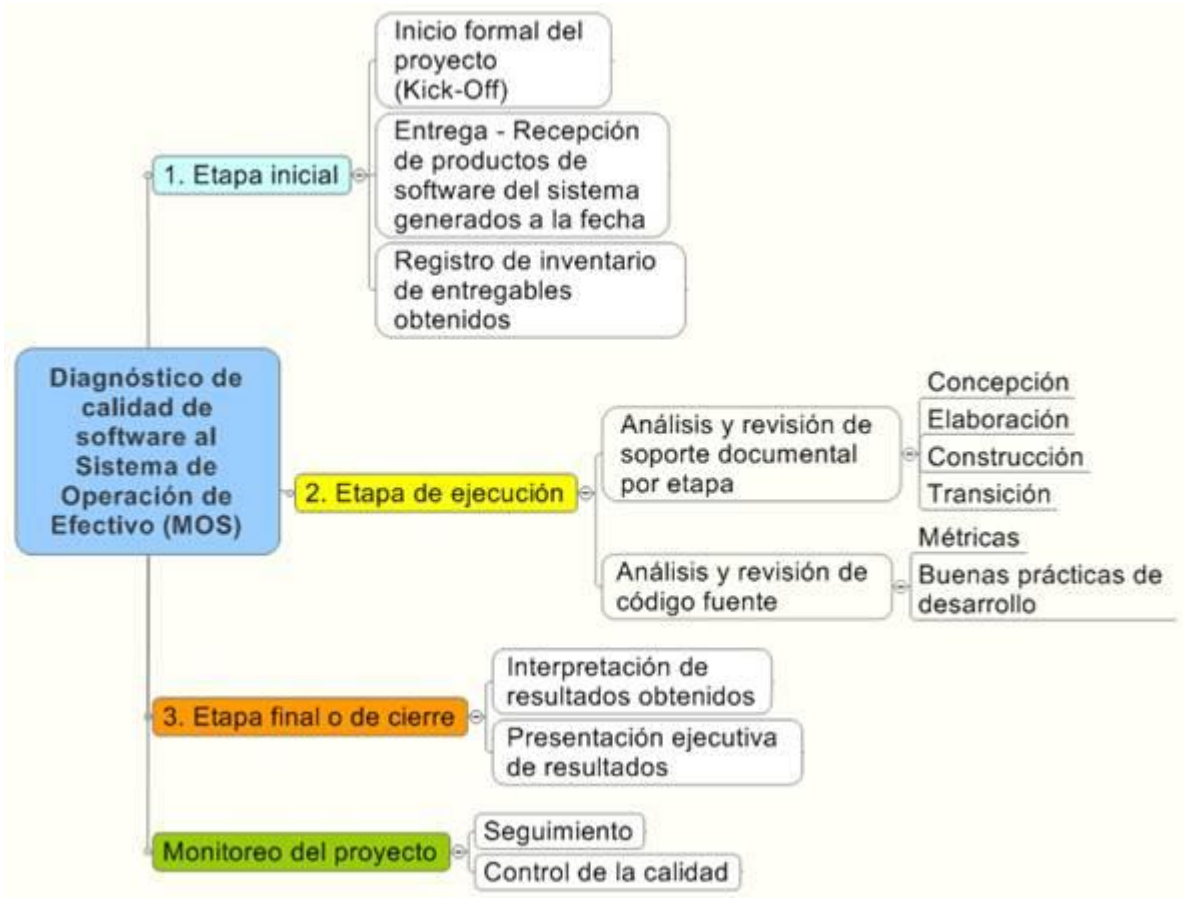


Figura 2 Estrategia general de ejecución de las actividades.

4.1 Etapa inicial

Después de establecer legalmente el proyecto con el cliente mediante un contrato de prestación de servicios, se comenzó con la etapa inicial de proyecto. Esta etapa tiene como propósito general formalizar las actividades y plan de proyecto, haciendo de su conocimiento a los involucrados con la finalidad de establecer un compromiso de su participación en cada una de ellas.

A continuación se describen de manera específica lo realizado en cada una de las actividades.

4.1.1 Reunión de inicio del proyecto

Esta reunión tiene como objetivo dar inicio formal a las actividades del proyecto, conferir responsabilidad a los involucrados en el proyecto, tanto por parte de los proveedores, como por parte del cliente. En la reunión se revisaron los siguientes elementos necesarios a considerar, para el buen desempeño de las actividades, haciendo de común acuerdo que se les daría seguimiento a lo largo del proyecto.

- Calendario del proyecto
- Plan de riesgos
- Recursos humanos asignados al proyecto
- Plan de comunicación
- Plan de datos
- Instalaciones y áreas de trabajo requeridas
- Dependencias
- Supuestos
- Herramientas.

4.1.2 Entrega/recepción de productos del sistema generados a la fecha

Esta actividad consistió en solicitar al cliente mediante una carta formal, toda la documentación generada a la fecha para el Módulo de Operación de Efectivo (MOS), dando ejemplos de lo que se podría considerar, tales como:

- **Documentación de contexto o ámbito del sistema**
- **Documentación derivada de la etapa de análisis del sistema** como especificación de requerimientos, casos de uso, diagramas de proceso
- **Documentación de diseño** como diagramas de secuencia, diagramas de clases o diseño *Entidad – Relación* de la base de datos
- **Documentación de la etapa de construcción**
- **Documentación de la etapa de implantación** del sistema generada a la fecha como manuales de usuario y manuales técnicos.

Al mismo tiempo se solicitó la normatividad y estándares utilizados para el desarrollo de sistemas por parte del cliente.

Posterior a la entrega de la carta de solicitud se recibió la documentación existente, la cual fue entregada en un disco compacto. Se formalizó la entrega firmando entre ambas partes una carta de entrega/recepción, que incluía la fecha en la que se recibió el disco así como el tamaño total físico en archivos que contenía.

4.1.3 Levantamiento de inventario de los entregables[1] obtenidos

Del disco compacto obtenido se procedió a verificar y clasificar los archivos contenidos en el mismo, identificando las versiones y fechas de cada uno de los documentos, así como del código fuente del sistema. Los documentos obtenidos se muestran en la sección de resultados de este informe en la sección de soporte documental (7.1).

Una vez levantado el inventario del contenido del disco compacto, se verificó con el cliente para validar la existencia de los archivos y la clasificación realizada, esto también sirvió para identificar los elementos que tomarían parte dentro del diagnóstico y aquellos que se excluirían debido a que no formaban parte de la documentación o productos de software que componen al sistema.

Finalmente se firmó de común acuerdo el inventario de entregables resultante y se entregó una copia al cliente.

4.2 Etapa de ejecución

Una vez que se contó con la documentación y código fuente del sistema, se procedió a realizar el diagnóstico de cada uno de los elementos obtenidos. Dentro de la documentación entregada se encontró un documento denominado “Normatividad para el desarrollo de sistemas”, la cual menciona que para el desarrollo de sistemas del cliente, es necesario utilizar el Proceso Unificado *Rational* (RUP, por sus siglas en inglés, Rational Unified Process) en su versión 1.3, el cual es un proceso de desarrollo definido que concibe cuatro etapas dentro del ciclo de desarrollo de un sistema de software, así como entregables definidos en cada etapa.

De manera general se mencionan a continuación las etapas que integran al RUP, una descripción general de ellas, y los criterios, consideraciones, herramientas y metodologías tomadas en cuenta para llevar a cabo el proceso de revisión de cada uno de los elementos que se encontraron en el inventario como productos derivados.

4.2.1 Fase de concepción

El objetivo de esta fase es establecer los requerimientos del cliente que cubrirá el sistema, identificando todas las entidades que interactúan con el sistema (personas, sistemas, etc.) y efectuar una valoración de la viabilidad del proyecto. Como productos mínimos esperados a encontrar está un documento de visión y ámbito del sistema, así como diagramas que muestren de manera general los procesos operativos del negocio.

Como criterio inicial para la revisión de esta etapa se identifica que se tengan los productos de trabajo mencionados, para posteriormente verificarlos mediante un proceso formal de

[1] Entregable no tiene traducción, pero por su definición es cualquier documento o producto que se entrega al cliente.

inspección con base en buenas prácticas y criterios del estándar del Lenguaje de Modelado Unificado (UML, por sus siglas en inglés, Unified Modeling Language). A continuación se listan las consideraciones tomadas en cuenta, que se buscó encontrar definidas en los documentos obtenidos para esta fase.

- Definición del objetivo que persigue la organización con la realización del sistema
- Delimitación de alcance
- Los beneficios que se obtendrán del sistema
- Definición de los procesos de negocio implicados en el sistema
- Reglas de negocio implicadas en el sistema
- Requerimientos de negocio susceptibles a satisfacer con funcionalidades que incluirá el sistema.

4.2.2 Fase de elaboración

El objetivo de esta fase es entender el problema desde el punto de vista del equipo de desarrollo. Lleva consigo la elaboración de la arquitectura del sistema y el diseño de la solución técnica, así como la determinación del plan del proyecto y la identificación de los riesgos fundamentales del mismo.

Como productos mínimos esperados se tienen los siguientes:

- Documento de casos de uso
- Modelo conceptual
- Diagrama de secuencias
- Diagrama de clases
- Diseño Entidad-Relación de la base de datos
- Diagrama general de la arquitectura de la solución.

4.2.3 Fase de construcción

En esta fase se profundiza en el diseño de los componentes y de manera iterativa se van añadiendo las funcionalidades al software a medida que se construyen y prueban, permitiendo a la vez que se puedan ir incorporando cambios.

La revisión en esta etapa consistió en un análisis con la herramienta *CodePro Analytix* de la compañía *Instantiations*, del código fuente en tres elementos principales:

- Métricas de código, con base en los factores de calidad definidos
- Análisis de código, tomando como referencia las buenas prácticas y estándares de programación Java

- Duplicidad de código, para encontrar elementos susceptibles a *refactorizar*[2].

Métricas de código

En este rubro la herramienta *CodePro Analytix* permite automatizar la obtención de métricas de código estándares, organizadas de manera tabular y relacionadas a un proyecto o paquete de código que se especifique. Cabe señalar que la herramienta despliega los valores obtenidos siendo responsabilidad del analista interpretar y obtener conclusiones de la información, como se mostrará en la sección de resultados (7). En la Figura 3 se muestra una imagen de la vista tabular de métricas obtenida para un paquete en particular.

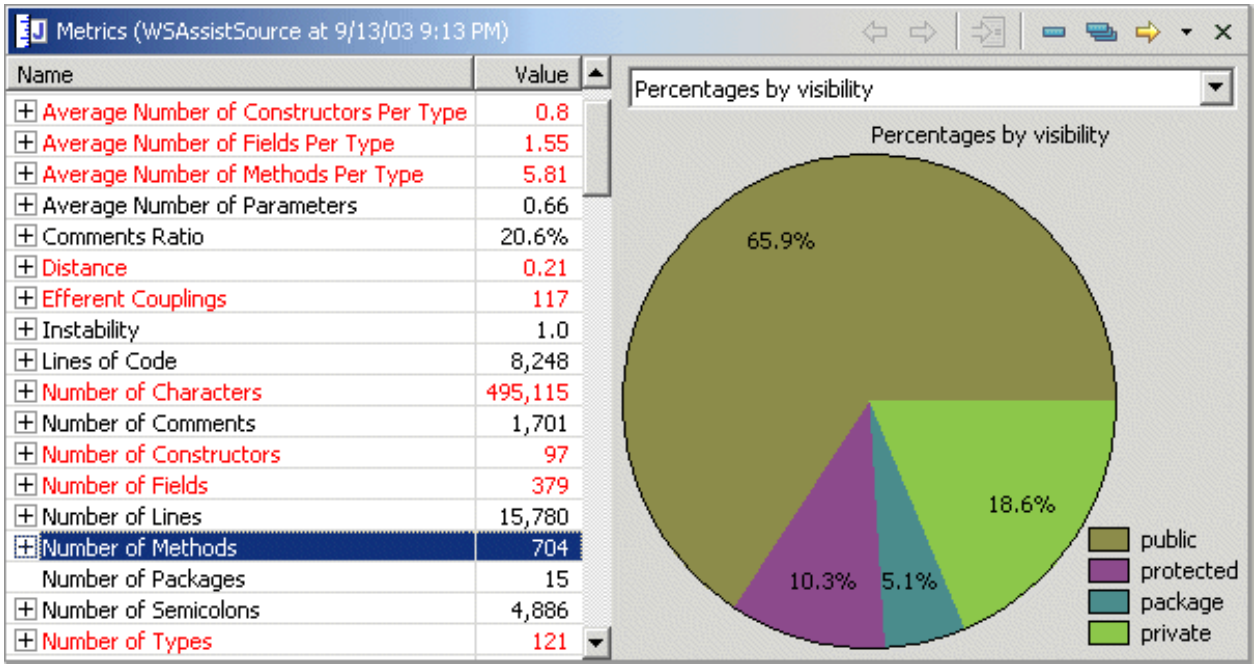


Figura 3 Vista tabular de resultados para la obtención de métricas de código con la herramienta CodePro Analytix.

[2] Refactorizar es modificar el código fuente sin cambiar su comportamiento para mejorar su consistencia interna y su claridad.

Análisis de código

Se detectan incidencias o defectos en unidades de código, tomando como referencia estándares de código predefinidos, reglas de seguridad y convenciones de estilo; la herramienta arroja un conjunto de 40 categorías de incidencias, sin embargo para fines prácticos del diagnóstico se consideraron únicamente las siguientes:

- Desempeño
- Complejidad de programa
- Código muerto o no alcanzable
- Portabilidad
- Estilo de código
- Convenciones de nomenclatura
- Manejo de excepciones
- Comentarios de código
- Seguridad.

Las Figuras 5 a 8 ejemplifican algunos reportes que se pueden obtener de la herramienta CodePro, mientras que la Figura 4 muestra un fragmento de código durante la ejecución de dicha herramienta.

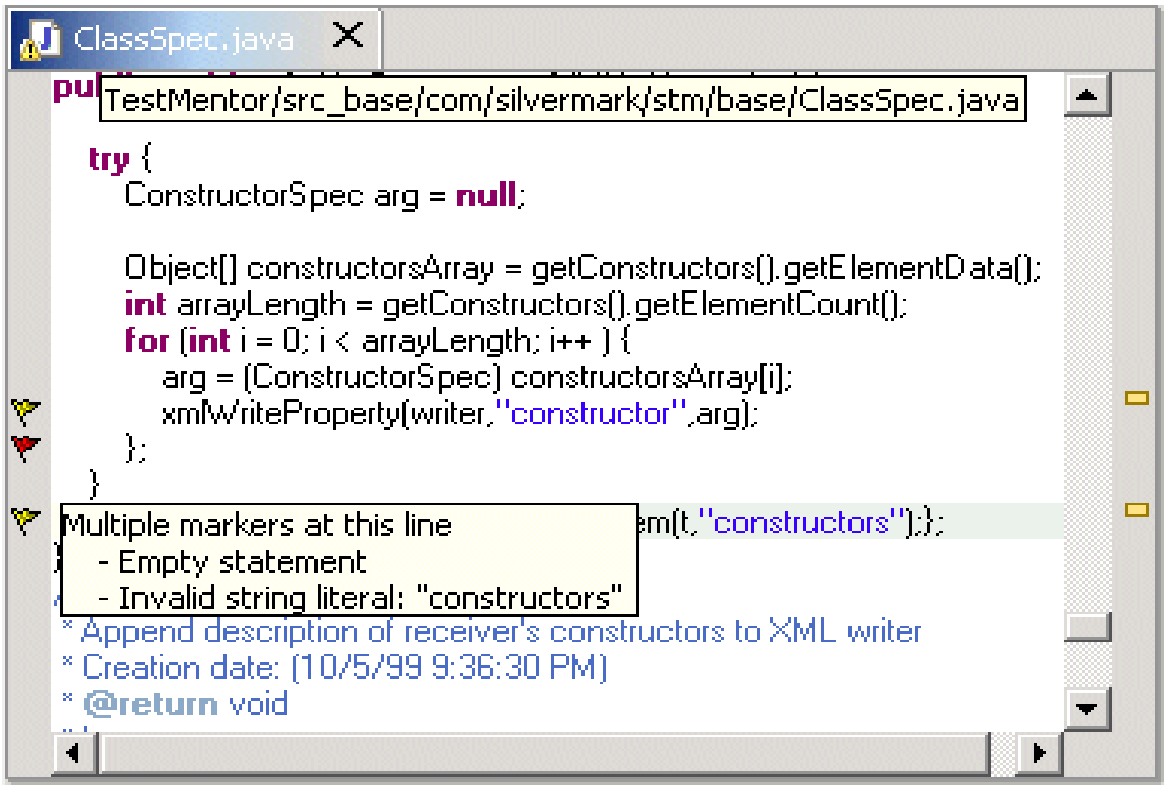


Figura 4 Vista de un fragmento de código de ejemplo donde se muestra cómo son detectadas las incidencias en el análisis del código.

Violations

Missing Javadoc comment for type Divisa	Opening brace for a type should appear on a new line	Opening brace for a method should appear on a new line
Line 15 is longer than 80 characters	Incorrect spacing between members	Opening brace for a method should appear on a new line
toString() is missing.	Incorrect spacing between members	Opening brace for a method should appear on a new line
readObject method missing	Incorrect spacing between members	Opening brace for a method should appear on a new line
The object or interface "Divisa" is declared as Serializable	Incorrect spacing between members	Opening brace for a method should appear on a new line
"Divisa" does not override clone()	Opening brace for a method should appear on a new line	Opening brace for a method should appear on a new line
writeObject method missing	Opening brace for a method should appear on a new line	

Source

```
1  /**
2   * Copyright (c) 2005-2006          All Rights Reserved.
3   */
4  package com.      .mos.persistence.model;
5
6  import java.io.Serializable;
7
8  import javax.persistence.Column;
9  import javax.persistence.Entity;
10 import javax.persistence.Id;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.Table;
13
14 @Entity
15 @Table(name = "C_DIVISA")
16 @NamedQuery(name = "Divisa.getDivisaByClaveAlfabetica", query = "FROM Divisa AS a WHERE a.claveAlfabetica = :claveAlfabetica")
17 public class Divisa implements Serializable{
18   /**
19    * serialVersionUID.
```

Figura 5 Ejemplo de incidencias de código encontradas en la clase Divisa por tipo de violación de código.

Violations

- Missing Javadoc comment for type EstadoCuenta
- toString() is missing.
- readObject method missing
- The object or interface "EstadoCuenta" is declared as Serializable
- "EstadoCuenta" does not override clone()
- writeObject method missing
- Opening brace for a type should appear on a new line
- Incorrect spacing between members
- Use of tab character outside a literal
- Incorrect spacing between members
- Incorrect spacing between members
- Opening brace for a method should appear on a new line
- Opening brace for a method should appear on a new
- Opening brace for a method should appear on a new
- Opening brace for a method should appear on a new
- Opening brace for a method should appear on a new
- Opening brace for a method should appear on a new

Source

```
1  /**
2   * Copyright (c) 2005-2006          All Rights Reserved.
3   */
4  package com.       .mos.persistence.model;
5
6  import java.io.Serializable;
7
8  import javax.persistence.Column;
9  import javax.persistence.Entity;
10 import javax.persistence.Id;
11 import javax.persistence.Table;
12
13 @Entity
14 @Table(name = "C_CUENTA_ESTADO")
15 public class EstadoCuenta implements Serializable {
16
17   /**
18    * serialVersionUID.
19    */
20   private static final long serialVersionUID = 1L;
21   /**
22    * Identificador del estado de la cuenta.
23    */
24   @Id
```

Figura 6 Ejemplo de incidencias de código encontradas en la clase Estado de Cuenta por tipo de violación de código.

Source

```
1  /**
2   * Copyright (c) 2005-2006          All Rights Reserved.
3   */
4  package com.       .mos.persistence.model;
5
6  /**
7   * Este catálogo contiene los distintos estados que puede tomar una instrucción
8   *
9   * @author aresendiz
10  * @since 1.0
11  */
12 public enum EstadoInstruccion {
13
14     /**
15      * Objeto EstadoInstruccion al que se le asigna el valor CON_MATCH_VALUE.
16      */
17     CON_MATCH("CON_MATCH", 0),
18
19     /**
20      * Objeto EstadoInstruccion al que se le asigna el valor SIN_MATCH_VALUE.
21      */
22     SIN_MATCH("SIN_MATCH", 1),
23     /**
24      * Objeto EstadoInstruccion al que se le asigna el valor CANCELADA_VALUE.
25      */
26     CANCELADA("CANCELADA", 2),
27     /**
28      * Objeto EstadoInstruccion al que se le asigna el valor PENDIENTE_VALUE.
29      */
30     PENDIENTE("PENDIENTE", 3),
31     /**
32      * Objeto EstadoInstruccion al que se le asigna el valor DEPENDIENTE_VALUE
```

Figura 7 Ejemplo de incidencias de código encontradas en la clase Instrucción por tipo de violación de código.

Violations

Missing Javadoc comment for type Mercado	Incorrect spacing between members	Incorrect spacing between members
toString() is missing.	Incorrect spacing between members	Opening brace for a method should appear on a new line
readObject method missing	Incorrect spacing between members	Incorrect spacing between members
The object or interface "Mercado" is declared as Serializable	Opening brace for a method should appear on a new line	Opening brace for a method should appear on a new line
"Mercado" does not override clone()	Incorrect spacing between members	Incorrect spacing between members
writeObject method missing	Opening brace for a method should appear on a new line	Opening brace for a method should appear on a new line
Opening brace for a type should appear on a new line	Incorrect spacing between members	
Incorrect spacing between members	Opening brace for a method should appear on a new line	

Source

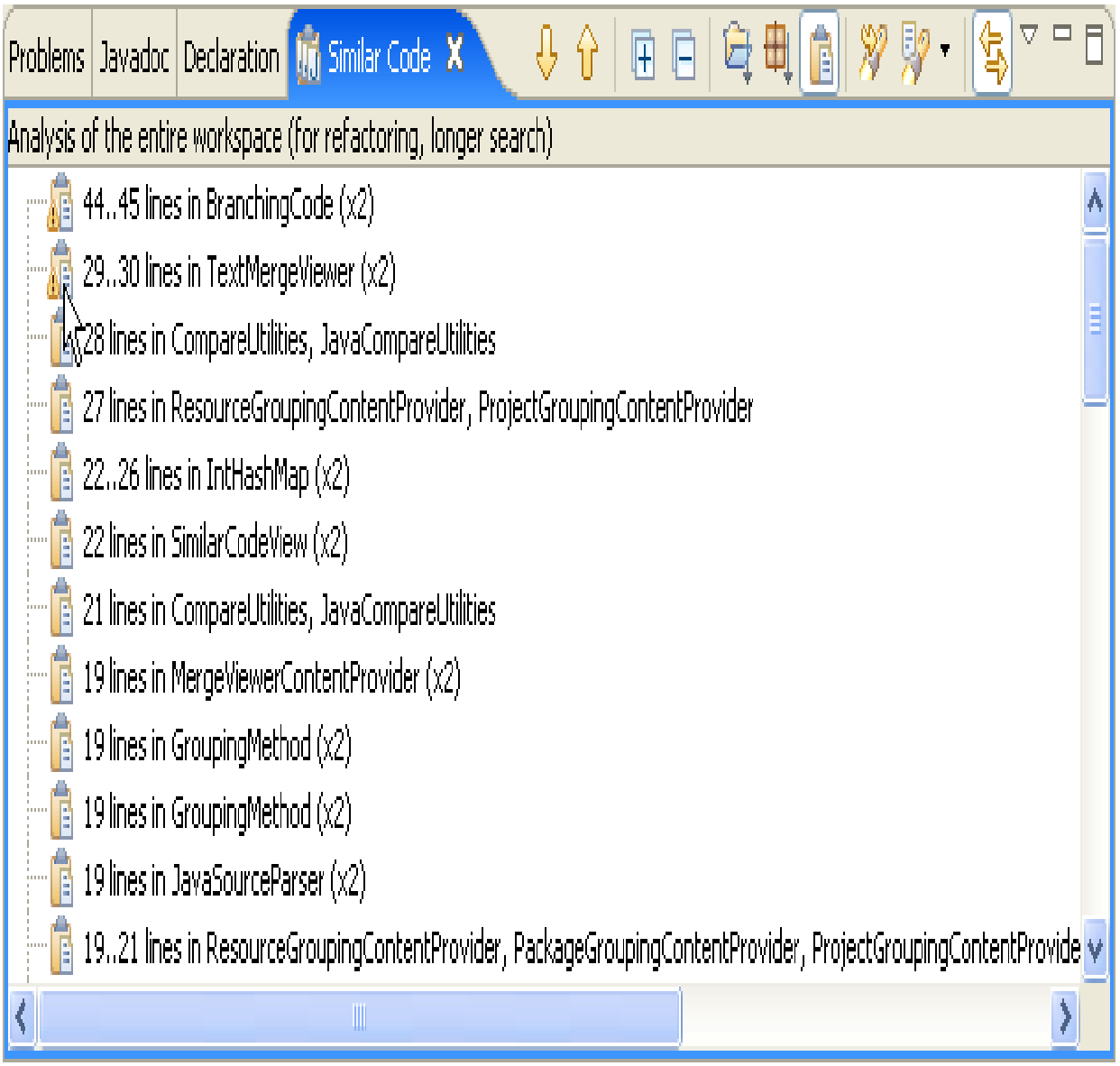
```
1  /**
2   * Copyright (c) 2005-2006          All Rights Reserved.
3   */
4  package com.      .mos.persistence.model;
5
6  import java.io.Serializable;
7
8  import javax.persistence.Column;
9  import javax.persistence.Entity;
10 import javax.persistence.Id;
11 import javax.persistence.Table;
12
13 @Entity
14 @Table(name = "C_MERCADO")
15 public class Mercado implements Serializable {
16
17   /**
```

Figura 8 Ejemplo de incidencias de código encontradas en la clase Mercado por tipo de violación de código.

Duplicidad de código

Se examina el código para identificar segmentos duplicados o muy similares, comúnmente como resultado de acciones copiar y pegar, lo que da muchas veces como consecuencia código extenso susceptible a *refactorizar* para mejorar el desempeño y su facilidad de mantenimiento.

En la Figura 9 se muestran algunas imágenes de ejemplo de la herramienta para poder localizar código duplicado.



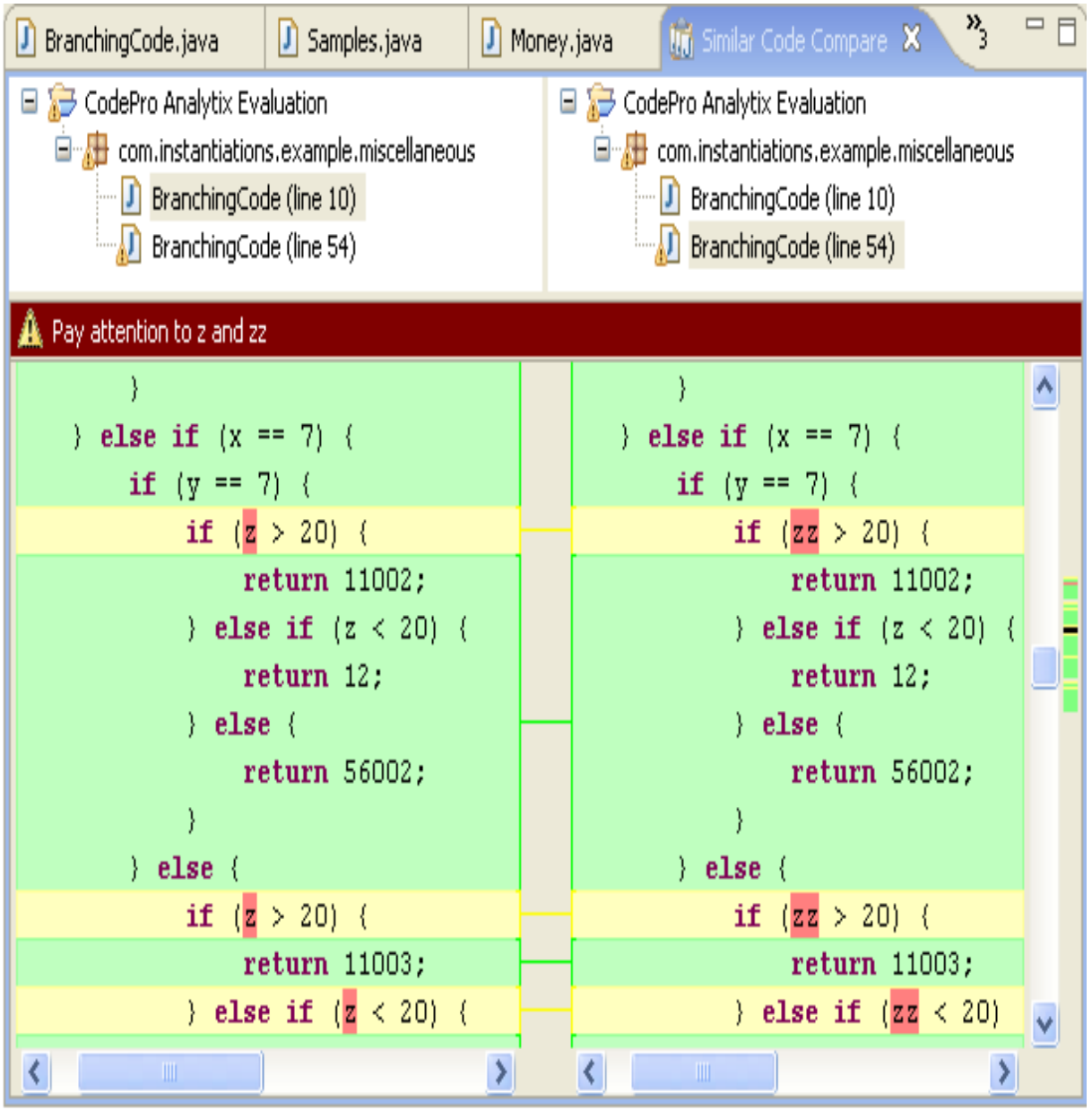


Figura 9 Imágenes de la herramienta CodePro para analizar código similar.

4.2.4 Fase de transición

La fase final del RUP se ocupa del traslado del software desde los entornos de desarrollo a los entornos de producción, en los que el usuario final hará uso del sistema. Dependiendo del tipo de proyecto, podrá requerir de entornos intermedios (preproducción o de aceptación por usuarios, etc.) para su correcta validación, antes de pasarlo a producción.

Esta etapa contempla los siguientes productos entregables.

- Manuales de usuario
- Manuales técnicos
- Guías de instalación
- Procedimientos para ajuste de desempeño de aplicación y puesta a punto.

Debido a que el sistema se encontraba aún en desarrollo, no se consideró la evaluación de dichos documentos ya que no estaban disponibles.

5 Análisis y metodología empleada

Al momento de definir la calidad de software se debe diferenciar entre la calidad del producto de software y la calidad del proceso de desarrollo de éste. No obstante, las metas que se establezcan para la calidad del producto van a determinar los objetivos a establecer de calidad del proceso de desarrollo, ya que la calidad del primero va a depender, entre otros aspectos, de ésta. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto.

Debido a que el sistema estaba en una etapa avanzada de su desarrollo, se llevó a cabo únicamente la revisión de calidad a productos y no al proceso de desarrollo; sin embargo, fueron considerados todos los artefactos[3] de software y códigos fuente que participaron en los estados de evolución del sistema como, especificaciones, diseños, códigos, manuales, entre otros.

La calidad del software es diferente a la calidad de otros productos de fabricación industrial, ya que el software tiene sus propias características específicas:

- El software es un producto mental, no restringido por las leyes de la física o por los límites de los procesos de fabricación. Es algo abstracto, un intangible.
- Se desarrolla, no se fabrica. El costo está fundamentalmente en el proceso de diseño, no en la posterior producción en serie, y los errores se introducen también en el diseño, no en la producción.
- Los costos del desarrollo de software se concentran en las tareas de ingeniería, mientras que en la fabricación clásica los costos se acentúan más en las tareas de producción.
- El software no se deteriora con el tiempo. No es susceptible de los efectos del entorno y su curva de fallos es muy diferente de la del hardware. Todos los problemas que surjan durante el mantenimiento estaban allí desde el principio y afectan a todas las copias del mismo; no se generan nuevos errores.
- El software, en su mayoría, se construye a la medida, en vez de ser construido ensamblando componentes existentes y ya probados, lo que dificulta aún más el control de su calidad.
- El mantenimiento del software es mucho más complejo que el mantenimiento del hardware. Cuando un componente del hardware se deteriora se sustituye por una pieza de repuesto, pero cada fallo en el software implica un error en el diseño o en el proceso mediante el cual se tradujo el diseño en código máquina ejecutable.
- Es engañosamente fácil realizar cambios sobre un producto de software, pero los efectos de estos cambios se pueden propagar de forma explosiva e incontrolada.
- Como disciplina, el desarrollo de software es aún muy joven, por lo que las técnicas de las que se dispone aún no están perfeccionadas.
- El software con errores no se rechaza. Se asume que es inevitable que el software presente algunos errores de menor impacto.

Para la revisión de productos se consideró como base de la revisión y marco de referencia el modelo de McCall, por ser uno de los más difundidos y porque además ha servido de base

[3] Artefacto es un producto tangible resultante del proceso de desarrollo de software.

para otros modelos como el modelo de Boehm y el Software Quality Management –SQM- de Murine.

En general los modelos de calidad definen a ésta de forma jerárquica, es decir, la calidad se produce como consecuencia de la evaluación de un conjunto de indicadores o métricas en diferentes etapas:

En el nivel más alto de la jerarquía se encuentran los factores de calidad definidos a partir de la visión del usuario del software, y conocidos también como atributos de calidad externos.

Cada uno de los factores se descompone en un conjunto de criterios de calidad, o sea aquéllos atributos que cuando están presentes contribuyen a obtener un software de la calidad. Se trata de una visión de la calidad técnica, desde el punto de vista del producto de software, y se les denomina también atributos de calidad internos.

Finalmente, para cada uno de los criterios de calidad se define un conjunto de métricas o medidas cuantitativas de ciertas características del producto, que indican el grado en que dicho producto posee un determinado atributo de calidad.

De esta manera, a través de un modelo de calidad se concretan los aspectos relacionados con ella de tal manera que se puede definir, medir y planificar el producto de software. Además el empleo de un modelo de calidad permite comprender las relaciones que existen entre diferentes características de un producto de software.

Un argumento de peso en contra de los modelos de calidad es que aún no ha quedado demostrada la validez absoluta de ninguno de ellos.

5.1 Modelo de McCall

El modelo de McCall organiza los factores de calidad de un sistema en tres ejes o puntos de vista desde los cuales el usuario puede contemplar la calidad de un producto, basándose en once factores de calidad organizados en torno a los tres ejes y a su vez cada factor se desglosa en otros criterios.

La Figura 10 ilustra este modelo a continuación.

Figura 10 Factores de calidad de un sistema de software – McCall 1997.

Revisión del producto

- **Facilidad de mantenimiento.** El esfuerzo requerido para localizar y reparar errores.
- **Flexibilidad.** El esfuerzo requerido para modificar una aplicación.
- **Facilidad de prueba.** El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos.

Operación del producto

- **Corrección.** El grado que una aplicación satisface sus especificaciones y consigue los objetivos encomendados.
- **Fiabilidad.** El grado que se puede esperar de una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida.
- **Integridad.** El grado con el que puede controlarse el acceso al software o a los datos a personal no autorizado.
- **Eficiencia.** El grado en el que una aplicación aprovecha los recursos de hardware y software para realizar sus operaciones con los tiempos de respuesta adecuados.
- **Facilidad de uso.** El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados.

Transición

- **Portabilidad.** El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo.
- **Reusabilidad.** Grado en que partes de la aplicación pueden utilizarse en otras aplicaciones.
- **Interoperabilidad[4].** El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos.

Antes de comenzar a utilizar el modelo de McCall hay que seguir las siguientes pautas:

1 Se aceptan los factores, criterios y métricas que propone el modelo.
2 Se aceptan las relaciones entre factores y criterios, y entre criterios y métricas.
3 Se selecciona un subconjunto de factores de calidad sobre los que se apliquen los requisitos de calidad establecidos para el proyecto.

Al comienzo del proyecto habrá que especificar los requisitos de calidad del producto de software, para lo cual se seleccionarán los aspectos inherentes a la calidad deseada del producto, teniendo que considerarse para ello:

- Las características particulares del propio producto que se está diseñando: por ejemplo, su ciclo de vida, que si se espera que sea largo, implicará un mayor énfasis en la facilidad de mantenimiento y la flexibilidad, o bien, si el sistema en desarrollo está destinado a un entorno donde el hardware evoluciona rápidamente, implicará como requisito su portabilidad.
- La relación calidad-precio, que puede evaluarse a través del costo de cada factor de calidad frente al beneficio que proporciona. La Tabla 1 muestra la relación calidad-precio para cada factor considerado para el sistema.

Tabla 1 Relación calidad-precio para cada factor.

Factor	Beneficio/Costo
Corrección	Alto
Fiabilidad	Alto
Eficiencia	Alto
Integridad	Alto
Facilidad de uso	Medio
Facilidad de mantenimiento	Alto
Facilidad de prueba	Alto
Flexibilidad	Medio
Portabilidad	Bajo
Reusabilidad	Bajo
Interoperabilidad	Alto

- La determinación de las etapas del ciclo de vida donde es necesario evaluar cada factor de calidad para conocer en cuáles se dejan sentir más los efectos de una calidad pobre

[4] Interoperabilidad es la habilidad que tiene un sistema para trabajar con otros sistemas sin un esfuerzo especial por parte del cliente.

con respecto a cada uno de los factores. Para el proyecto se consideró evaluar todas las etapas con sus artefactos del ciclo de vida del proceso unificado, estas son:

1 Concepción
2 Elaboración
3 Construcción
4 Transición.

- Las propias interrelaciones entre los factores, debido a que algunos factores pueden entrar en conflicto entre sí: por ejemplo, la eficiencia plantea conflictos prácticamente con todos los demás factores de calidad. La interacción entre los diversos factores a evaluar queda reflejada en la Figura 11 que indica la dependencia entre los factores de McCall.

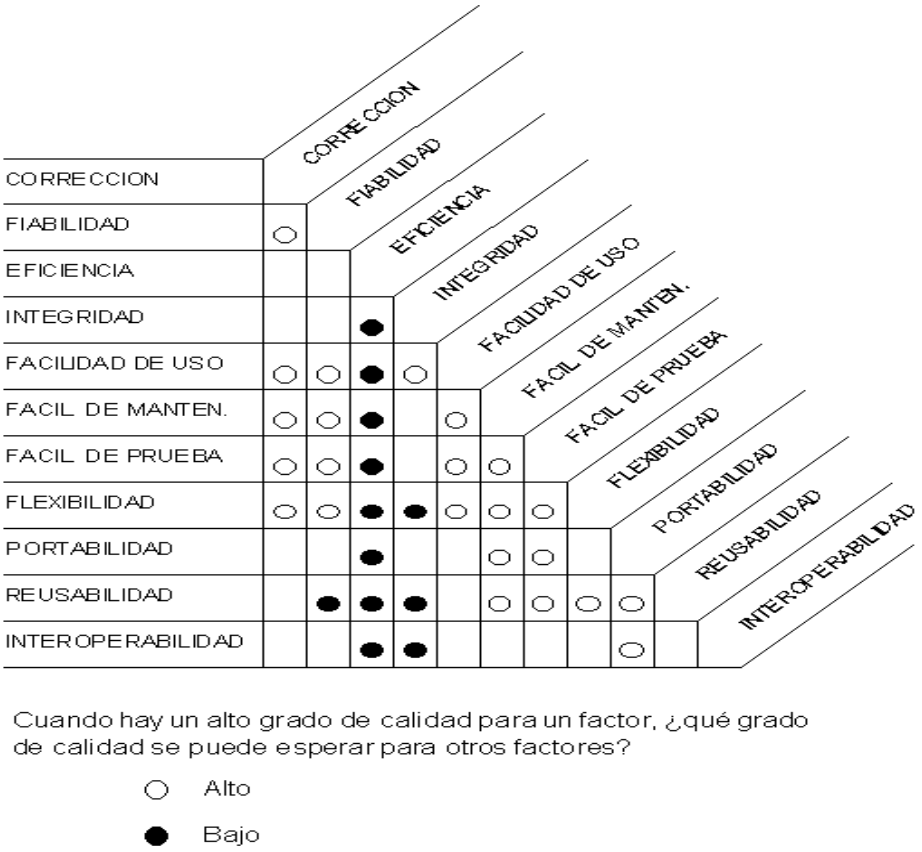


Figura 11 Relación entre factores de calidad.

5.2 Métricas de código

Las métricas de software son indicadores que permiten conocer la medida de algún parámetro específico dentro de un sistema de software. Las métricas actuales y los modelos están lejos de la perfección, pero aplicados apropiadamente, pueden proveer mejoras muy significativas en el proceso de desarrollo de software.

La administración de proyectos de software es actualmente poco efectiva ya que el desarrollo de software es muy complejo y se tienen pocas formas de medir los procesos o productos para guiar y evaluar el desarrollo. Por lo tanto, la estimación, el control y la planeación efectivos y precisos son casi imposibles de lograr. Así que para mejorar el proceso de administración, es necesario tener la habilidad de identificar, medir y controlar los parámetros esenciales del proceso de desarrollo. Éste es el objetivo de las métricas del software, la identificación y medición de los parámetros esenciales que afectan al desarrollo de software.

Se han propuesto y utilizado métricas y modelos de software por algún tiempo. Sin embargo, las métricas no han sido utilizadas en forma regular y metódica. Los resultados recientes indican que la implementación y aplicación concienzuda de un programa de métricas de software puede ayudar a lograr mejores resultados de administración, tanto a corto plazo para un proyecto dado como a largo plazo mejorando la productividad en proyectos futuros.

Las métricas deben facilitar la creación de modelos que sean capaces de predecir parámetros de procesos o productos, no solo describirlos. Por lo tanto, las características recomendables que se quieren encontrar en las métricas son:

- **sencillas**, para que sea claro cómo puede ser evaluada dicha métrica
- **objetivas**, lo más que sea posible
- **fácilmente calculadas** u obtenidas (es decir, a un costo razonable)
- **válidas**, la métrica debe medir lo que su diseño dice que debe medir
- **robustas**, relativamente insensibles a los cambios insignificantes en el proceso o producto.

5.2.1 Clasificación de las métricas

Las métricas pueden clasificarse en *métricas de producto* y *métricas de proceso*. Las métricas de producto son mediciones del producto de software en cualquier estado de su desarrollo, desde los requerimientos hasta el sistema instalado. Las métricas de producto pueden medir la complejidad del diseño, el tamaño del programa final, etc. Las métricas de proceso, por otro lado, son mediciones del proceso de desarrollo de software, tal como tiempo total de desarrollo, o el nivel promedio de experiencia del personal de programación.

En el caso particular de este proyecto se concentrará en la definición de las métricas de producto, ya que es un sistema ya construido el cual se requiere diagnosticar.

5.2.1.1 Métricas de tamaño

Las métricas de tamaño permiten cuantificar de manera objetiva qué tan grande o pequeño puede ser un sistema, dando una primera aproximación de la complejidad del mismo. Las siguientes son algunas de las métricas diseñadas para medir el tamaño del software.

Número de líneas. Se refiere al conteo bruto de líneas ignorando las líneas con espacios en blanco o vacías.

Número de líneas de código. Indica el conteo de número de líneas de código sin considerar comentarios ni líneas con espacios en blanco o vacías.

Número de clases. Número total de clases e interfaces definidas.

Número de constructores. Número de métodos constructores en clases e interfaces.

Número de métodos. Número de operaciones implementadas en las clases e interfaces.

Número de atributos. Número de atributos clases e interfaces.

Número de paquetes. Número total de paquetes utilizados en el sistema.

Número de comentarios. Conteo del número de comentarios, los comentarios que contienen líneas múltiples son contabilizados como un solo comentario.

Razón de comentarios por líneas de código. Relación entre el total de líneas de comentarios y el total de líneas de código, que indica la proporción de comentarios por cada línea de código.

Número de casos de uso. Conteo del número de casos de uso identificados en la etapa de análisis del sistema.

Número de tablas. Número de entidades o tablas establecidas en el diseño de la base de datos.

Número de atributos. Número de atributos en las entidades o tablas establecidas en el diseño de la base de datos.

5.2.1.2 Métricas de complejidad

Estas son algunas de las métricas que se han propuesto para medir la complejidad de los programas. Así como con las métricas de tamaño, las mediciones de complejidad pueden ser medidas desde las etapas iniciales de la construcción de un sistema. Dentro de las métricas de complejidad se tienen:

Profundidad de bloque. Esta métrica identifica métodos y constructores que tienen niveles de bloques anidados. Un método con varios niveles de bloques anidados puede ser difícil de entender.

Complejidad *ciclomática*. [5]La complejidad ciclomática en un método es la medida del número de distintas rutas de ejecución dentro del método. Las diferentes rutas que puede seguir un método están dadas por las instrucciones de control *if, else, case, catch,* y los operadores condicionales *&& y* ||. En la Tabla 2 se muestra la relación entre complejidad ciclomática y el riesgo que representa darle mantenimiento en un método.

Tabla 2 Relación entre complejidad ciclomática y riesgo.

Complejidad ciclomática	Riesgo
1 a 10	Métodos simples, riesgo bajo.
11 a 20	Métodos más complejos, riesgo moderado.
21 a 50	Métodos complejos, riesgo alto.
>50	Métodos inestables, riesgo muy alto.

Peso de métodos por clase. Esta métrica es la suma de complejidades de los métodos definidos en una clase. Por consiguiente, representa la complejidad de una clase como un todo y esta medida se puede utilizar para indicar el esfuerzo de desarrollo y mantenimiento de la clase.

Nivel de profundidad por tipo. Número total de los antepasados de una clase (directa o indirecta) en la jerarquía de herencia.

5.2.1.3 Métricas de programación orientada a objetos

Las métricas para sistemas de programación orientada a objetos (mejor conocidos por sus siglas OO) deben ajustarse a las características que distinguen el software OO del software convencional. Estas métricas deben tomar en cuenta características como encapsulamiento, herencia, polimorfismo, ocultamiento de información y técnicas de abstracción.

Uno de los conjuntos de métricas de programación orientada a objetos al que se hace más referencia es el propuesto por Chidamber y Kemener (CK) que incluye:

[5] Complejidad ciclomática se refiere a la medición cuantitativa de la complejidad lógica condicional de un programa.

a Métodos por clase
b Nivel de profundidad
c Número de descendientes
d Respuesta para una clase
e Falta de cohesión.

En el libro de métricas realizado por Lorenz y Kidd hay cuatro categorías de métricas: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaño para una clase OO se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema OO en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones a lo largo y ancho de la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.

Las métricas de Robert C. Martin se usan para medir la calidad de un diseño OO en términos de la interdependencia de sus subsistemas. Los diseños con alta interdependencia tienden a ser rígidos, no reutilizables y difíciles de mantener, pero cierta interdependencia es necesaria. Las métricas creadas para este fin son:

a Ca (*Afferent*[6] *Couplings*)
b Ce (*Efferent*[7] *Couplings*)
c I – Inestabilidad
d A – Abstracción.

Las métricas que aplicamos en este sistema fueron:

Ca (Afferent Couplings). Número de clases fuera del paquete que dependen de las clases dentro del paquete.

Ce (Efferent Couplings). Número de clases dentro del paquete que dependen de las clases fuera del paquete.

I Inestabilidad. (Ce / (Ca + Ce)). Esta métrica esta en el rango [0,1]. I=0 indica una categoría con máxima estabilidad. I=1 indica una categoría con máxima inestabilidad.

A Abstracción. (Número de clases abstractas en la categoría / número total de clases en la categoría). Esta métrica está en el rango [0,1]. 0 significa concreta y 1 significa completamente abstracta.

[6] ,
[7] Las palabras afferent y efferent no tienen traducción pero por definición implican una relación de acoplamiento hacia adentro en el primer caso, y de acoplamiento hacia afuera en el segundo.

Líneas de código por método. Número de líneas de código utilizadas en la implementación de métodos de todas las clases entre la cantidad total de métodos.

Constructores por clase. Número de métodos constructores en el sistema entre el número total de clases**.**

Atributos por clase. Número de atributos implementados en todas las clases.

Métodos por clase. Número de métodos implementados en todas las clases.

Número de parámetros. Relación entre el número de parámetros utilizados en todos los métodos implementados en el sistema dividida entre el número total de métodos.

Falta de cohesión. Establece en qué medida los métodos hacen referencia a los atributos. Valores cercanos a cero indican que la mayoría de los métodos acceden a la mayoría de las instancias, en cambio valores altos indican falta de cohesión, proponiendo que la clase está compuesta de elementos no relacionados.

6 Participación profesional

El rol desempeñado dentro del proyecto fue el de Ingeniero de aseguramiento de la calidad de productos de software, en el cual implicó tener conocimiento de los conceptos y técnicas de calidad de software, así como identificar las propiedades de calidad que deben cumplir los productos para poder llevar a cabo su verificación.

El grupo completo del proyecto de aseguramiento de la calidad del módulo de operación de efectivo estuvo integrado por un líder de proyecto, dos ingenieros de aseguramiento de la calidad y un ingeniero de pruebas.

Dentro de las actividades desempeñadas por el rol de ingeniero de aseguramiento de la calidad, se tienen las siguientes:

- Realizar planes de actividades de aseguramiento de la calidad junto con el líder de proyecto
- Llevar a cabo el inventario de los entregables obtenidos del cliente
- Realizar la inspección de productos de software para comprobar su calidad, como:
    - Documentos de especificaciones de requerimientos
    - Documentos de arquitectura del sistema
    - Documentos de diseño del sistema
    - Diagrama Entidad-Relación de la base de datos
    - Documentación de código fuente
    - Manuales de usuario
    - Manuales técnicos
    - Planes de proyecto
    - Calendario de proyecto
    - Documentos de pruebas
    - Documentos de implantación del sistema, como guías de configuración e instalación
- Obtener y registrar métricas de software
- Realizar análisis e interpretación de métricas de software
- Participar en reuniones de presentación de resultado al cliente
- Registrar bitácoras de actividades realizadas
- Dar seguimiento a la corrección de incidencias o defectos identificados
- Preparar informes de resultados de revisiones
- Construir presentaciones de resultados para el cliente.

Los entregables que fueron responsabilidad del rol, se listan a continuación:

- Inventario de software obtenido del cliente
- Informe de resultados de inspecciones a productos
- Informe de métricas de software

- Informe final de calidad del producto
- Plan de acción para la adecuación de incidencias a corto, mediano y largo plazos.

Los trabajos de revisión del módulo de operación de efectivo tuvieron una duración de diez semanas, desde el inicio formal del proyecto hasta la presentación de resultados al cliente.

- Informe final de calidad del producto
- Plan de acción para la adecuación de incidencias a corto, mediano y largo plazos.

Los trabajos de revisión del módulo de operación de efectivo tuvieron una duración de diez semanas, desde el inicio formal del proyecto hasta la presentación de resultados al cliente.

7 Resultados y aportaciones

A continuación se describen los resultados encontrados en la documentación y código fuente del Módulo de Operación de Efectivo. Con base en el proceso de revisión y diagnóstico de los sistemas de software de la compañía, se obtuvieron los resultados divididos en las siguientes categorías:

1 Revisión a soporte documental
2 Métricas de código
3 Estándares y buenas prácticas en código fuente.

La forma en cómo se presentan los resultados es: primero, dando una breve introducción del proceso para llevar a cabo la revisión en particular; posteriormente, se listan las incidencias o no conformidades encontradas a las que se les denominó hallazgos, y tienen los siguientes atributos:

- Identificador del hallazgo
- Elemento al que hace referencia
- Severidad
- Descripción
- Recomendaciones.

7.1 Revisión a soporte documental

La revisión al soporte documental se llevó a cabo mediante un proceso formal de inspección de software, con la finalidad identificar defectos u omisiones en los entregables del sistema MOS. Las etapas que se contemplan en el proceso son las siguientes:

1 Planificación de las actividades de revisión
2 Revisión de estándares y normas para el desarrollo de aplicaciones de software
3 Preparación para la revisión
4 Revisión detallada
5 Registro de no conformidades y observaciones.

Se consideraron como elementos a revisar, los especificados en el apartado de elementos de configuración considerados para la revisión de este documento. Así también se tomó como referencia para realizar las inspecciones, los estándares y guías para el desarrollo de sistemas.

Como resultado de la revisión de cada documento se encuentran los hallazgos y recomendaciones para cada incidencia a las que se hace referencia mediante un identificador único.

7.1.1 Elementos considerados para la revisión

- **Documento de arquitectura del Módulo de Operación de Efectivo (MOS)** (versión 1.5 con fecha última de actualización del 7 de diciembre del 2007). Este documento contiene la descripción del diseño y la arquitectura general de la interfaz con el módulo Liquidador Efectivo del Banco de México. También explica los detalles de diseño a partir de diferentes vistas, donde cada vista es una descripción del sistema desde un aspecto de diseño distinto.

- **Documentación HTML generada del Módulo de Operación de Efectivo** (última fecha de publicación del 27 de mayo del 2008). Esta documentación está integrada por tres secciones: la primera sección “Módulo de Operación de Efectivo” contiene diagramas de flujo de depósitos, retiros y traspasos de efectivo, así como la especificación de los mensajes utilizados y ejemplos de éstos; la segunda sección “Módulos” contiene una lista de seis módulos, sin embargo, la liga a cada uno de estos marca error debido a que no se encuentra la página a la que se hace referencia; la tercera sección “Documentación del Proyecto” contiene la información del proyecto y reportes del proyecto, dicha documentación fue generada automáticamente por la herramienta Maven (la cual se describe en la sección 7.1.2) tomando todos los documentos y ligas que forman parte del proyecto.

7.1.2 Referencias a estándares y guías utilizadas en la construcción de la documentación

El área de desarrollo de sistemas del cliente utiliza como guía el documento mencionado a continuación, por lo que la revisión de calidad se llevó a cabo con base en este mismo documento, con la finalidad de cumplir con los mismos estándares.

Documento de arquitectura general el desarrollo de un nuevo sistema, versión 1.2, 27 de octubre del 2007.

Este documento describe el proceso de desarrollo a utilizar para la construcción de nuevos sistemas, el cual es el RUP; y norma que se debe utilizar UML como lenguaje de modelado y descripción de sistemas. También define los entregables que se deben generar en cada etapa del proceso de desarrollo.

El documento también incluye otras herramientas que deben utilizarse para el desarrollo de sistemas, que son:

- Jira, que es una herramienta de software para el seguimiento de defectos y errores
- Maven, que es una herramienta para la gestión y construcción de proyectos en Java
- *Modelo Vista Controlador,* que es un patrón de arquitectura de software que separa los datos, interfaz de usuario, y la lógica de control en tres componentes distintos.

7.1.3 Hallazgos y recomendaciones

A continuación se describen los hallazgos y observaciones encontradas en la revisión e inspección de cada unos de los artefactos que conforman el soporte documental del módulo. La organización de dichos hallazgos es de acuerdo al proceso de desarrollo utilizado, el cual es denominado "*Proceso Unificado*"; éste considera las siguientes etapas:

1 Concepción
2 Elaboración
3 Construcción
4 Transición.

Esta sección clasifica los hallazgos según su severidad, utilizando un código de colores: el color azul indica una severidad baja, el color amarillo indica una severidad media y el color rojo indica una severidad alta.

Concepción

Hallazgo 1	
Elemento	Documentación de visión y ámbito del sistema
Severidad	Baja
Descripción	No se cuenta con un documento específico que describa el planteamiento del sistema desde el punto de vista del negocio
Recomendación	Se sugiere realizar un documento que especifique la visión y ámbito del negocio para la realización del sistema incluyendo entre otros puntos los siguientes: -Objetivo de negocio -Alcance -Beneficios -Procesos de negocio -Reglas de negocio -Requerimientos de negocio.

Hallazgo 2	
Elemento	Diagramas de modelado del negocio
Severidad	Baja
Descripción	No se cuenta con los diagramas que permitan conocer o entender los procesos operativos del negocio que debe implementar el sistema
Recomendación	Realizar al menos un diagrama general del modelo de negocio o hacer referencia si ya se cuenta con el mismo.

Hallazgo 3	
Elemento	Documento de casos de uso
Severidad	Baja
Descripción	No se tiene un vínculo o relación de los casos de uso del sistema con los requerimientos de negocio
Recomendación	Establecer un documento de los casos de uso del sistema relacionados a los requerimientos de negocio definidos.

Elaboración

Hallazgo 4	
Elemento	Especificaciones de definición y diseño del MOS en documento de arquitectura
Severidad	Media
Descripción	Se identificaron faltas, omisiones o errores de definición y diseño en el documento de arquitectura del MOS, como se lista a continuación: – Se especifica que hay una funcionalidad del sistema para recepción de instrucciones del PF (Protocolo Financiero), siendo que ésta es la única funcionalidad descrita para recepción, y se recibe de otros módulos también como el módulo del portal y la interfaz con el liquidador de efectivo; se debería quitar la leyenda de PF o agregar las demás funcionalidades de recepción para los demás módulos. – Existen validaciones que no especifican códigos de error devueltos, se utilizan comodines o valores no definidos como XXXX
Recomendación	Revisar y ajustar las observaciones listadas.

Hallazgo 5	
Elemento	Documento de especificación de requerimientos de software
Severidad	Media
Descripción	No se cuenta con el documento de especificación de requerimientos de software
Recomendación	Elaborar documento como lo describe el documento de arquitectura general del desarrollo de un nuevo sistema.

Hallazgo 6	
Elemento	Especificación de casos de uso en documento de arquitectura
Severidad	**Alta**
Descripción	Se identificaron las siguientes incidencias en la especificación de casos de uso, las cuales se pueden ver reflejadas como omisiones o falta de apego al estándar UML actual [Unified Modeling Language Specification™ (UML®) ver. 1.4.2, www.omg.org]

	– El diagrama de casos de uso no integra completamente los elementos estándar de un diagrama, omitiendo los límites del sistema, identificación de actores primarios y secundarios. – Se identifica como actor principal al MOS, siendo éste el sistema a modelar en el diagrama de casos de uso. – Los casos de uso identificados no describen tareas del negocio sino estos pueden ser más bien requerimientos funcionales del sistema como “Procesar instrucciones”, “Validar instrucciones”, “Expirar instrucciones”, entre otras. – Los casos de uso identificados se plasma como una secuencia de actividades o pasos, cuando esto debe hacerse en los escenarios de un caso de uso. Por ejemplo, “Recepción de instrucciones, percepción de instrucciones, aplicar reglas de negocio, ejecución de procesos, etc.” – No se identifican ni especifican escenarios alternos ni excepcionales para los casos de uso. – En las precondiciones se plasmaron las validaciones que realiza el sistema, mismas que inclusive se pueden considerar como reglas de negocio. – No se tienen especificados los escenarios de los casos de uso de forma tal que muestren la interacción entre actores como una secuencia de pasos. – No se tienen vinculados los requerimientos técnicos o funcionales a los casos de uso.
Recomendación	Adecuar los casos de uso con base en las observaciones realizadas.

Hallazgo 7	
Elemento	Secuencias
Severidad	Media
Descripción	Se encontraron las siguientes no conformidades en la especificación de los diagramas de secuencia, las cuales se pueden ver reflejadas como omisiones o falta de apego al estándar UML actual [Unified Modeling Language Specification™ (UML®) ver. 1.4.2, www.omg.org] – No se identifican los actores de forma específica. – No se especifican los pasos del flujo de eventos del caso de uso a modelar
Recomendación	Adecuar los diagramas de secuencia de acuerdo a las observaciones realizadas.

Hallazgo 8	
Elemento	Modelo conceptual
Severidad	Baja
Descripción	No se cuenta con un modelo conceptual que permita establecer las entidades
Recomendación	Elaborar diagrama conceptual del sistema.

Hallazgo 9	
Elemento	Diagrama de clases
Severidad	Alta
Descripción	No se cuenta con un diagrama de clases que sirva como referencia del diseño de clases del sistema
Recomendación	Elaborar diagrama de clases.

Hallazgo 10	
Elemento	Diagrama Entidad-Relación de la base de datos
Severidad	Alta.
Descripción	No se cuenta con un diagrama Entidad-Relación de la base de datos que permita conocer la relación, normalización y la cardinalidad de las entidades
Recomendación	Elaborar diagrama Entidad-Relación indicando la cardinalidad y relación entre entidades hasta tercera forma normal de acuerdo al diseño del sistema.

Hallazgo 11	
Elemento	Diccionario de la base de datos
Severidad	Alta
Descripción	No se cuenta con el diccionario de datos definido que explique el propósito de cada tabla y columna
Recomendación	Elaborar diccionario de datos

Construcción

Hallazgo 12	
Elemento	Documentación Javadoc
Severidad	**Alta**
Descripción	No se cuenta con la documentación de código fuente Javadoc
Recomendación	Generar la API de la documentación del código fuente.

Hallazgo 13	
Elemento	Documentación de pruebas
Severidad	Media
Descripción	No se cuenta con evidencia documental de haber realizado pruebas del sistema, sin embargo parte de los códigos fuentes integran las clases de prueba
Recomendación	Si se realizaron pruebas, sería recomendable tener los resultados de las mismas.

Hallazgo 14	
Elemento	Documentación de despliegue
Severidad	Media
Descripción	No se cuenta con la documentación de despliegue que facilite la instalación y configuración de la aplicación
Recomendación	Realizar documentación como matriz de configuración de código fuente, procedimientos de integración de código fuente a ambiente de desarrollo y ambientes de producción.

Transición

No aplicó, debido a que el proyecto se encontraba en la fase final de construcción y no se contaba aún con ningún tipo de documento para esta fase.

Observaciones generales

Hallazgo 15	
Elemento	Control de versiones
Severidad	Media
Descripción	No se establece un control de versiones con administración de líneas base que permita garantizar la integridad de la documentación
Recomendación	Establecer un mecanismo de control de versiones.

Hallazgo 16	
Elemento	Actualización de la documentación
Severidad	Alta
Descripción	No se tiene actualizada la documentación existente con los últimos cambios realizados
Recomendación	Actualizar la documentación.

7.1.4 Resumen de incidencias

En la Figura 12 se muestran las incidencias clasificadas por su severidad. En esta gráfica podemos apreciar que tenemos el mismo número de incidencias con severidad alta que con severidad media y el número de incidencias con severidad baja es ligeramente menor.

Figura 12 Incidencias a adecuar por severidad.

En la Figura 13 podemos ver que durante la etapa de elaboración se incurrió en un número mayor de incidencias, mientras que en las etapas de concepción y construcción solamente hubieron tres incidencias en cada una.

Figura 13 Incidencias a adecuar por etapa.

7.2 Métricas de código

Con la última versión entregada del Módulo de Operación de Efectivo (MOS) identificada con fecha del 27 de mayo del 2008, se analizó el código con dos herramientas diferentes para la obtención de un conjunto de métricas, las cuales se clasificaron en tres grupos, métricas de tamaño de la aplicación, métricas de complejidad y métricas de programación orientada a objetos.

Cabe mencionar que la segunda herramienta utilizada fue *Metrics*, pero los resultados y la evaluación estuvieron basados en la herramienta *CodePro*, por lo que sólo se hace referencia a ésta última en el informe.

Una vez obtenido el valor de las diferentes métricas se especificaron un conjunto de hallazgos y observaciones preliminares de los resultados obtenidos que sirvieron como base para establecer las recomendaciones y sugerencias de ajuste para garantizar con un buen nivel de confianza la estabilidad del sistema.

7.2.1 Métricas de tamaño del sistema

Tabla 3 Métricas de tamaño del sistema.

Métrica	Valor
Líneas de código físicas	16,561
Líneas de código lógicas (SLOC)	7,622
Número de clases	122
Número de métodos	1,066
Promedio de métodos por clase	8.73
Número de atributos	688
Número de constructores	27
Número de paquetes	41
Número de comentarios	1,825
Razón de comentarios con respecto a líneas de código	23.9%
Número de tablas de BD	16
Número de atributos de BD	114

Observaciones y hallazgos de métricas de tamaño del sistema

1 De acuerdo al número de líneas de código, número de clases y número de métodos de la lógica de la aplicación, se pudo establecer que el módulo de “Operación de Efectivo”, es un sistema de software de tamaño pequeño, basándose en la clasificación hecha por el ANSI (por sus siglas en inglés, American National Standards Institute) en una revisión extensa de proyectos Java (ver Tabla 4).

Tabla 4 Líneas de código (SLOC) vs. tamaño del sistema.

Líneas de código	Tamaño
0 a 9,999	Pequeño
10,000 a 49,000	Medio
50,000 a 99,000	Entre medio y grande
100,000 a 499,999	Grande
>500,000	Muy grande

2 Se pudo observar de las métricas que el número de constructores es inferior al número de clases. Es correcto que las clases hijas no implementen constructor en el caso de que ya lo definan las clases padres; sin embargo, convendría revisar las clases que no sean heredadas y que se haya omitido su constructor por defecto, con la finalidad de implementarlo.

3 La razón de comentarios con respecto a las líneas de código obtenido (23.9%) permitió establecer que el código tiene un nivel de cobertura documental adecuado (se recomienda de 20 a 30%). Sin embargo, se encontraron casos en donde los comentarios pueden no ser tan descriptivos o completos.

4 Se justificó que se haya tenido un número pequeño de tablas y campos debido a que únicamente se consideraron aquellas entidades en la base de datos de las cuales deja registro el módulo de Operación de Efectivo.

7.2.2 Métricas de complejidad del sistema

En la Tabla 5, la Tabla 6, la Tabla 7, y la Tabla 8 se muestran los resultados de las métricas de complejidad del sistema, y posteriormente se incluyen las observaciones correspondientes.

Tabla 5 Métricas de complejidad del sistema.

Métrica	Media	Mínimo	Máximo	Desv.Est.	Observaciones
Profundidad de bloque	0.72	0	6	0.53	Tres es el valor máximo recomendado
Complejidad ciclomática (CC)	1.23	1	37	1.71	Ver tabla de complejidad ciclomática
Peso de métodos por clase	12.04	1	104	9.71	100 como valor máximo recomendado
Nivel de profundidad por tipo	2.24	1	5	0.48	Diez es el valor máximo recomendado.

Tabla 6 Otras métricas de complejidad del sistema.

Métrica	Valor
Número de métodos largos	7
Número de constructores largos	1
Número de inicializadores largos	0
Número de clases con demasiados constructores	0
Número de clases con demasiados atributos	16
Número de clases con demasiados métodos	28
Número de métodos con demasiados atributos	3
Número de unidades largas	0

Observaciones y hallazgos de métricas de complejidad del sistema

1 El valor medio de la **profundidad de bloque** permitió establecer que la mayoría de los métodos de la aplicación están por debajo del valor máximo recomendado tres (3) con una varianza baja; sin embargo, existen seis clases que salen del valor recomendado. La Tabla 7 lista estas clases.

Tabla 7 Lista de clases que salen del valor recomendado de profundidad de bloque.

Clase	Profundidad de bloque
AdaptadorMessageProcessorImpl.java	5
InterfazMessageProcessorImpl.java	4
SlvMessageProcessorImpl.java	5
MosValidationServiceImpl.java	4
ResponseMessageServiceImpl.java	6
XStreamFactory.java	4

2 La **complejidad ciclomática** en su valor medio (1.23) indica que dicha métrica se mantiene en un valor estable al considerar el total de métodos de la aplicación, pero por otra parte se puede identificar en paquetes y clases por separado que la métrica sale del valor recomendado. La lista de clases complejas con riesgo moderado (seis clases) y riesgo alto (cuatro clases) se muestra en la Tabla 8.

Tabla 8 Lista de clases complejas con riesgo moderado y riesgo alto.

Clase	Método	Máximo de CC
	Riesgo Moderado	
AdaptadorMessageProcessorImpl.java	Process	15
InterfazMessageProcessorImpl.java	Process	17
BitacoraServiceImpl.java	guardaMensajeAdaptador	15
ResponseMessageServiceImpl.java	buildMensajeSLVbyAdaptadorMessage	12
XStreamFactory.java	XStreamFactory	11
EstadoInstruccion.java	EstadoInstruccion	14
	Riesgo Alto	
MosValidationServiceImpl.java	validaMensajePago	23
MosValidationServiceImpl.java	validaMensajeTraspaso202	37
MosValidationServiceImpl.java	validaMensajeTraspaso203	24
UtilDate.java	validaFecha	29

7.2.3 Métricas de programación orientada a objetos

En esta sección se describen los resultados obtenidos en aquellas métricas que se relacionan con la implementación del código tomando como base las buenas prácticas de codificación de la ingeniería de software y los principios de programación orientada a objetos.

Tabla 9 Métricas de programación orientada a objetos.

Métrica	Valor	Observaciones
Inestabilidad	0.28	El rango es entre 0 y 1; entre más cercano al cero más estable
Dependencias de las clases dentro del paquete hacia otras clases fuera del paquete (Efferent Couplings Ce)	3.33	20 es el valor máximo recomendado
Dependencias de otras clases fuera del paquete hacia las clases dentro del paquete (Afferent Coupling Ca)	8.63	20 es el valor máximo recomendado
Abstracción	15.5%	Niveles de abstracción grandes permiten una mayor reusabilidad del código
Falta de cohesión	0.37	Rango entre 0 y 1; cerca del 1 implica falta de cohesión.

Tabla 10 Otras métricas de programación orientada a objetos.

Métrica	Media	Mínimo	Máximo	Desv. Est.	Observaciones
Constructores por clase	0.22	0	2	0.09	Al menos uno por clase
Líneas de código por método	5.04	1	126	8.76	30 es el valor máximo recomendado
Atributos por clase	4.92	0	49	2.08	10 es el valor máximo recomendado
Métodos por clase	8.73	0	66	4.78	10 es el valor máximo recomendado
Parámetros por método	0.71	0	7	0.71	Seis es el valor máximo recomendado.

Tabla 11 Incidencias con código similar o duplicado.

Métrica	Valor	Observaciones
Identificar código duplicado para *refactorizar*	1	Se encontró una incidencia, se sugiere su revisión
Identificar código copiado con posibles errores de renombrado	2	Se encontraron dos incidencias de métodos similares que posiblemente fueron copiados, se sugiere su revisión
Identificar código que parece similar	41	Se encontraron 41 incidencias de métodos similares, se sugiere su revisión.

Observaciones y hallazgos de métricas de programación orientada a objetos

1 Los valores de las métricas orientadas a objetos se comportaron de manera estable. Sin embargo, se recomienda revisar las clases que salen de los valores recomendados de atributos, parámetros, constructores y métodos.

2 Se encontraron siete clases que salen del valor máximo de líneas de código por método recomendado (30).

Tabla 12 Lista de clases que salen del número de líneas de código por método recomendado.

Clase	Método	Observaciones
MensajePago103Vo.java	MensajePago103Vo	80% más de lo recomendado (30 recomendado, 54 encontrados)
MensajeTraspaso202Vo.java	MensajeTraspaso202Vo	47% más de lo recomendado (30 recomendado, 44 encontrados)
CuentaNombrada.java	CuentaNombrada	7% más de lo recomendado (30 recomendado, 32 encontrados)
InstruccionEfectivo.java	InstruccionEfectivo	77% más de lo recomendado (30 recomendado, 53 encontrados)
InstruccionLiquidacion.java	InstruccionLiquidacion	30% más de lo recomendado (30 recomendado, 39 encontrados)
OperacionNombrada.java	OperacionNombrada	120% más de lo recomendado (30 recomendado, 66 encontrados)
TipoOperacion.java	TipoOperacion	47% más de lo recomendado (30 recomendado, 44 encontrados).

3 Se encontró una clase que sale del valor máximo de líneas por constructor recomendado (30).

Tabla 13 Lista de clases que exceden el número de líneas por constructor recomendado.

Clase	Constructor	Observaciones
XStreamFactory.java	XStreamFactory	33% más de lo recomendado (30 es el valor recomendado y se encontraron 40 líneas).

4 Se encontraron 16 clases que exceden el valor máximo de atributos por clase recomendado diez (10).

Tabla 14 Lista de clases que exceden el número de atributos recomendado.

Clase	Observaciones
MosConstantes.java	390% más de lo recomendado (10 recomendado, 49 encontrados)
MensajePago103Vo.java	160% más de lo recomendado (10 recomendado, 26 encontrados)
MensajeTraspaso202Vo.java	110% más de lo recomendado (10 recomendado, 21 encontrados)
MensajeTraspaso203Vo.java	10% más de lo recomendado (10 recomendado, 11 encontrados)
TraspasosVo.java	20% más de lo recomendado (10 recomendado, 12 encontrados)
CuentaControlada.java	10% más de lo recomendado (10 recomendado, 11 encontrados)
CuentaNombrada.java	60% más de lo recomendado (10 recomendado, 16 encontrados)
EstadoInstruccion.java	50% más de lo recomendado (10 recomendado, 15 encontrados)
InstruccionEfectivo.java	150% más de lo recomendado (10 recomendado, 25 encontrados)
InstruccionLiquidacion.java	90% más de lo recomendado (10 recomendado, 19 encontrados)
OperacionNombrada.java	230% más de lo recomendado (10 recomendado, 33 encontrados)
PosicionOperacionNombrada.java	10% más de lo recomendado (10 recomendado, 11 encontrados)

RegContEfecControlada.java	20% más de lo recomendado (10 recomendado, 12 encontrados)
RegContValControlada.java	10% más de lo recomendado (10 recomendado, 11 encontrados)
TipoCuenta.java	10% más de lo recomendado (10 recomendado, 11 encontrados)
TipoOperacion.java	130% más de lo recomendado (10 recomendado, 23 encontrados).

5 Se encontraron 28 clases que exceden el valor máximo de métodos recomendado diez (10).

Tabla 15 Lista de clases que exceden el número de métodos recomendado.

Clase	Observaciones
BitacoraService.java	10% más de lo recomendado (10 recomendado, 11 encontrados)
BitacoraServiceImpl.java	90% más de lo recomendado (10 recomendado, 19 encontrados)
MosValidationServiceImpl.java	20% más de lo recomendado (10 recomendado, 12 encontrados)
ResponseMessageServiceImpl.java	130% más de lo recomendado (10 recomendado, 23 encontrados)
MensajeConfirmacionVo.java	110% más de lo recomendado (10 recomendado, 21 encontrados)
MensajeISO950Vo.java	60% más de lo recomendado (10 recomendado, 16 encontrados)
MensajePago103Vo.java	440% más de lo recomendado (10 recomendado, 54 encontrados)
MensajeTraspaso202Vo.java	340% más de lo recomendado (10 recomendado, 44 encontrados)
MensajeTraspaso203Vo.java	140% más de lo recomendado (10 recomendado, 24 encontrados)
TraspasosVo.java	140% más de lo recomendado (10 recomendado, 24 encontrados)
BitacoraMensajeEntrada.java	20% más de lo recomendado (10 recomendado, 12 encontrados)
CuentaControlada.java	120% más de lo recomendado (10 recomendado, 22 encontrados)
CuentaNombrada.java	220% más de lo recomendado (10 recomendado, 32 encontrados)
Emision.java	80% más de lo recomendado (10 recomendado, 18 encontrados)

Institucion.java	80% más de lo recomendado (10 recomendado, 18 encontrados)
InstruccionEfectivo.java	430% más de lo recomendado (10 recomendado, 53 encontrados)
InstruccionLiquidacion.java	290% más de lo recomendado (10 recomendado, 39 encontrados)
Instrumento.java	40% más de lo recomendado (10 recomendado, 14 encontrados)
OperacionNombrada.java	560% más de lo recomendado (10 recomendado, 66 encontrados)
PosicionNombrada.java	80% más de lo recomendado (10 recomendado, 18 encontrados)
PosicionOperacionNombrada.java	120% más de lo recomendado (10 recomendado, 22 encontrados)
RegContEfecControlada.java	130% más de lo recomendado (10 recomendado, 23 encontrados)
RegContValControlada.java	110% más de lo recomendado (10 recomendado, 21 encontrados)
SaldoControlada.java	60% más de lo recomendado (10 recomendado, 16 encontrados)
SaldoNombrada.java	80% más de lo recomendado (10 recomendado, 18 encontrados)
TipoCuenta.java	100% más de lo recomendado (10 recomendado, 20 encontrados)
TipoInstruccion.java	40% más de lo recomendado (10 recomendado, 14 encontrados)
TipoOperacion.java	340% más de lo recomendado (10 recomendado, 44 encontrados).

6 Se encontraron tres clases que sobrepasan el valor máximo de parámetros por método recomendado seis (6).

Tabla 16 Lista de clases que exceden el número de parámetros por método recomendado.

Clase	Método	Observaciones
BitacoraService.java	guardaMensajeTraspasoSalida	17% más de lo recomendado (seis recomendado, siete encontrados)
RegContEfecControlada.java	RegContEfecControlada	34% más de lo recomendado (seis recomendado, ocho encontrados)
RegContValControlada.java	RegContValControlada	34% más de lo recomendado (seis recomendado, ocho encontrados).

7.3 Análisis de código fuente

A continuación se muestran los hallazgos encontrados en la auditoría de código fuente. Éstos se organizaron en tres categorías: aquellas que son factibles de adecuar a un corto, mediano o largo plazos. Los criterios utilizados para clasificar las incidencias en dichas categorías fueron los siguientes:

Corto plazo. Incidencias que pueden solventarse en un periodo de dos a tres semanas y mejoran factores como desempeño, fiabilidad, facilidad de mantenimiento, de preferencia en niveles de severidad altos.

Mediano plazo. Incidencias que se pueden resolver en un periodo de cuatro semanas a dos meses y su nivel de severidad está en medio o bajo.

Largo plazo. Incidencias a justificar o ajustar en el periodo de mantenimiento de la aplicación.

7.3.1 Incidencias factibles de adecuar a corto plazo

Desempeño

No. 1	**Código de depuración en versión en producción**
Ocurrencias	**16**
Severidad	**Alta**
Descripción	Instrucciones comúnmente utilizadas en código en depuración se encontraron en código en producción
Factor de calidad	Desempeño
Categoría	Desempeño
Recomendación	Eliminar líneas de código.

No. 2	**Método invocado en condición de un ciclo**
Ocurrencias	**3**
Severidad	**Alta**
Descripción	La invocación de método en la condición de un ciclo lo forzará a ser ejecutado al menos tantas veces como se ejecute el ciclo
Factor de calidad	Desempeño
Categoría	Desempeño
Recomendación	Invocar el método antes del ciclo, a no ser que el método devuelva un valor diferente, cada vez que es llamado.

No. 3	Métodos no usados
Ocurrencias	5
Severidad	Media
Descripción	Se implementó el método, pero no es usado en ninguna parte del programa
Factor de calidad	Desempeño
Categoría	Código muerto
Recomendación	Eliminar método.

No. 4	Variables declaradas dentro de un ciclo
Ocurrencias	17
Severidad	Baja
Descripción	Se encontraron variables declaradas dentro de un ciclo
Factor de calidad	Desempeño
Categoría	Desempeño
Recomendación	Revisar código para declarar las variables fuera del ciclo.

Fiabilidad

No. 5	Omisión de código en instrucción if
Ocurrencias	1
Severidad	Alta
Descripción	No se implementó código en instrucción if; esto puede dar lugar a un error, en caso de recibir un valor que cumpla la condición del if
Factor de calidad	Fiabilidad
Categoría	Posibles errores
Recomendación	Implementar o quitar instrucción if.

Mantenimiento

No. 6	Código comentado
Ocurrencias	24
Severidad	Media
Descripción	Se encontraron líneas de código comentadas
Factor de calidad	Facilidad de mantenimiento
Categoría	Comentarios
Recomendación	Eliminar líneas de código.

Resumen de incidencias

Por factor de calidad

En la Figura 14 podemos ver que la más del 50% de las incidencias podría afectar el desempeño del sistema y aunque son menos, las incidencias que están relacionadas con el mantenimiento, representan una cantidad considerable.

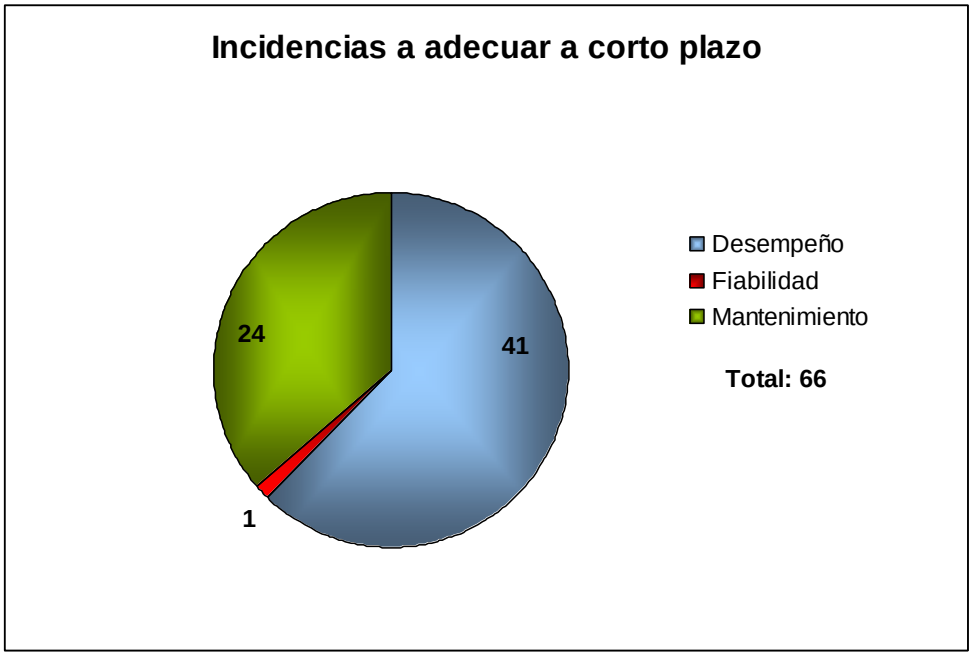


Figura 14 Incidencias a adecuar a corto plazo

Por severidad

En la Figura 15 podemos ver que las incidencias con severidad media son las más abundantes, pero también tenemos un número considerable de incidencias con severidad alta y con severidad baja.

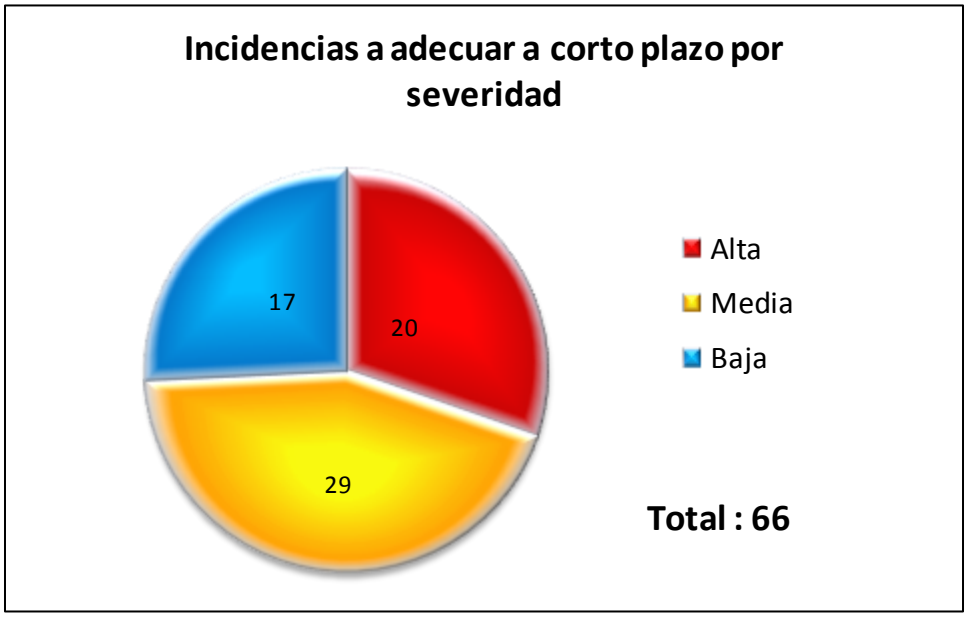


Figura 15 Incidencias a adecuar a corto plazo por severidad.

7.3.2 Incidencias factibles de adecuar a mediano plazo

Fiabilidad

No. 7	**Manejo de excepciones de forma muy general**
Ocurrencias	**16**
Severidad	**Media**
Descripción	Manejo de excepciones de forma muy general, dificultando su recuperación
Factor de calidad	Fiabilidad
Categoría	Manejo de excepciones
Recomendación	Realizar el manejo de excepciones de forma más especifica.

Portabilidad

No. 8	**Separador de línea específico de la plataforma**
Ocurrencias	**40**
Severidad	**Media**
Descripción	Se utilizan separadores de líneas “\n” en cadenas que pueden hacer a la aplicación dependiente de la plataforma en donde se compila
Factor de calidad	Portabilidad
Categoría	Portabilidad
Recomendación	Eliminar separadores de línea.

Mantenimiento

No. 9	**Profundidad de bloque**
Ocurrencias	**6**
Severidad	**Media**
Descripción	Se encontraron bloques con niveles de anidamiento altos
Factor de calidad	Mantenimiento
Categoría	Otros
Recomendación	Revisar código para reducir niveles de anidamiento.

No. 10	**Complejidad ciclomática.**
Ocurrencias	**10**
Severidad	**Media**
Descripción	Se encontraron métodos con riesgo moderado y alto debido a su complejidad
Factor de calidad	Mantenimiento y desempeño
Categoría	Otros
Recomendación	Revisar métodos y separar código en otros métodos.

No. 11	Uso de literales numéricas
Ocurrencias	72
Severidad	Baja
Descripción	Se encontraron números en los nombre de variables
Factor de calidad	Mantenimiento
Categoría	Otros
Recomendación	Omitir números en la declaración de variables.

Resumen de incidencias

Por factor de calidad

En la Figura 16 se puede ver que las incidencias relacionadas con el mantenimiento son las que tuvieron más ocurrencias, seguidas por aquellas relacionadas con la portabilidad y por último las incidencias relacionadas con la fiabilidad.

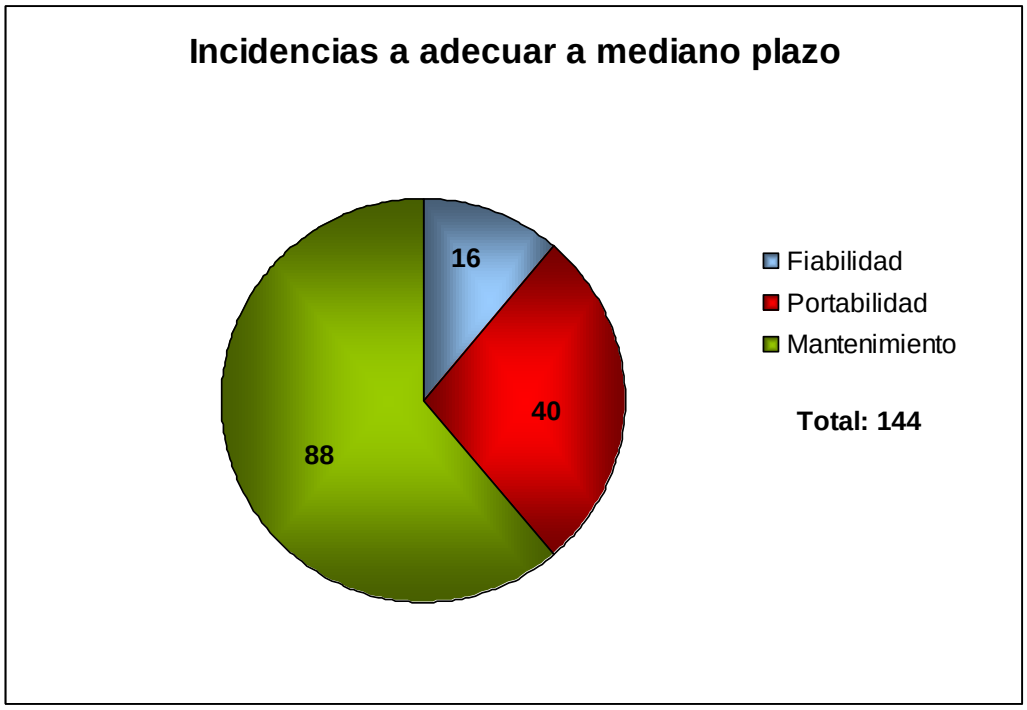


Figura 16 Incidencias a adecuar a mediano plazo.

Por severidad

En la Figura 17 vemos como resultaron igual número de incidencias con severidad media y baja y no hubieron incidencias con severidad alta.

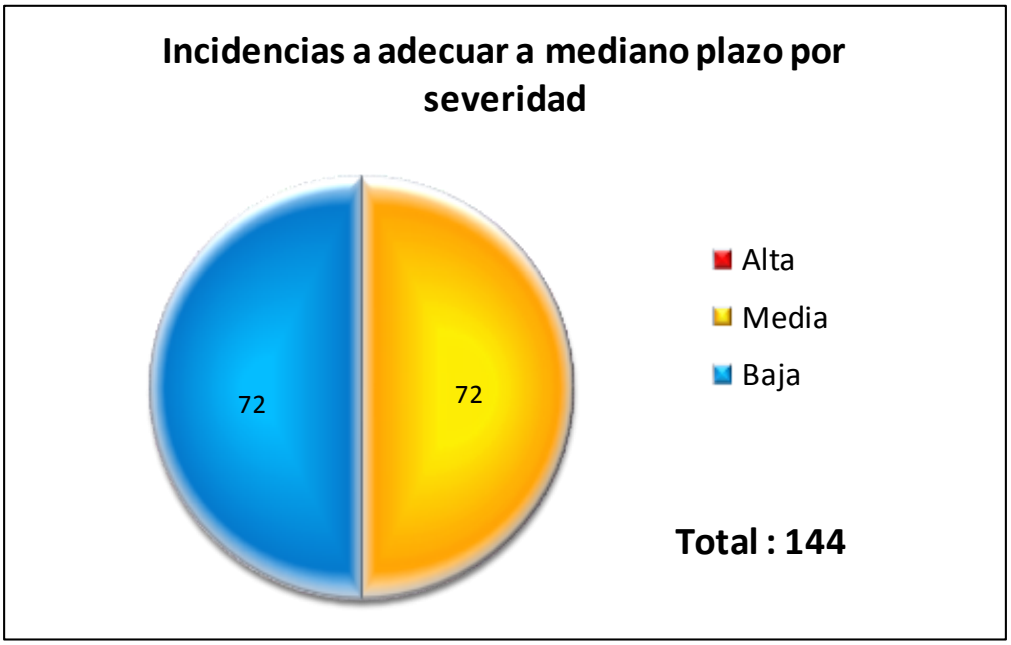


Figura 17 Incidencias a adecuar a mediano plazo por severidad.

7.3.3 Incidencias factibles de adecuar a largo plazo

Mantenimiento

No. 12	**Omisión de comentarios Javadoc para atributos de clase**
Ocurrencias	**210**
Severidad	**Media**
Descripción	No se incluye en los comentarios Javadoc a nivel de clase la especificación para variables
Factor de calidad	Facilidad de mantenimiento
Categoría	Convenciones Javadoc
Recomendación	Agregar definición de variables en comentarios Javadoc.

No. 13	**Omisión de etiqueta @param para comentario Javadoc**
Ocurrencias	**142**
Severidad	**Media**
Descripción	No se incluye la definición de parámetros para métodos en la etiqueta @param en el comentario Javadoc
Factor de calidad	Facilidad de mantenimiento
Categoría	Convenciones Javadoc
Recomendación	Agregar definición de parámetros en comentarios Javadoc.

No. 14	**Omisión de etiqueta @version para comentarios Javadoc**
Ocurrencias	**69**
Severidad	**Media**
Descripción	No se incluye la definición de etiqueta de @version en comentarios Javadoc
Factor de calidad	Facilidad de mantenimiento
Categoría	Convenciones Javadoc
Recomendación	Agregar definición de versión en comentarios Javadoc.

No. 15	**Uso no recomendado de operador condicional “: ?”**
Ocurrencias	**44**
Severidad	**Baja**
Descripción	El uso del operador condicional “: ?” puede causar confusión en mantenimiento
Factor de calidad	Facilidad de mantenimiento
Categoría	Estilo de codificación
Recomendación	Reemplazar operador condicional por instrucciones if, else.

No. 16	**Uso de identificador this no necesario**
Ocurrencias	**18**
Severidad	**Baja**
Descripción	Uso de identificador this en métodos de clase que no es necesario
Factor de calidad	Facilidad de mantenimiento
Categoría	Estilo de codificación
Recomendación	Eliminar identificador.

No. 17	**Falta de paréntesis en operadores condicionales**
Ocurrencias	**35**
Severidad	**Baja**
Descripción	No se utilizan paréntesis junto con el operador condicional
Factor de calidad	Facilidad de mantenimiento
Categoría	Estilo de codificación
Recomendación	Incorporar paréntesis en operador.

No. 18	**Variables locales no inicializadas o no declaradas al principio del bloque**
Ocurrencias	**156**
Severidad	**Baja**
Descripción	No se inicializó una variable o declaró al principio de un método
Factor de calidad	Facilidad de mantenimiento
Categoría	Estilo de codificación
Recomendación	Inicializar variables y declararlas al principio del método.

No. 19	**Convención de nombrado para métodos booleanos**
Ocurrencias	**12**
Severidad	**Baja**
Descripción	Se recomienda utilizar un prefijo estándar para el nombrado de métodos booleanos
Factor de calidad	Facilidad de mantenimiento
Categoría	Convenciones de nombrado
Recomendación	Definir un estándar y adecuar nombre de métodos booleanos.

No. 20	**Nombre de constantes no válidas**
Ocurrencias	**4**
Severidad	**Baja**
Descripción	Se encontraron declaración de constantes con números
Factor de calidad	Facilidad de mantenimiento
Categoría	Convenciones de nombrado
Recomendación	Definir un estándar y adecuar nombre de constantes.

No. 21	Convención de nombre de atributos no válido
Ocurrencias	9
Severidad	Baja
Descripción	No se utilizó el estándar para el nombrado de atributos de clase
Factor de calidad	Facilidad de mantenimiento
Categoría	Convenciones de nombrado
Recomendación	Adecuar nombre de atributos con base en el estándar.

No. 22	Nombre de enumeraciones no válidas
Ocurrencias	10
Severidad	Baja
Descripción	Se encontraron declaración de enumeraciones con números
Factor de calidad	Facilidad de mantenimiento
Categoría	Convenciones de nombrado
Recomendación	Definir un estándar y adecuar nombre de enumeraciones.

Resumen de incidencias

Por factor de calidad

En la Figura 18 se observa que todas las incidencias están relacionadas con el mantenimiento.

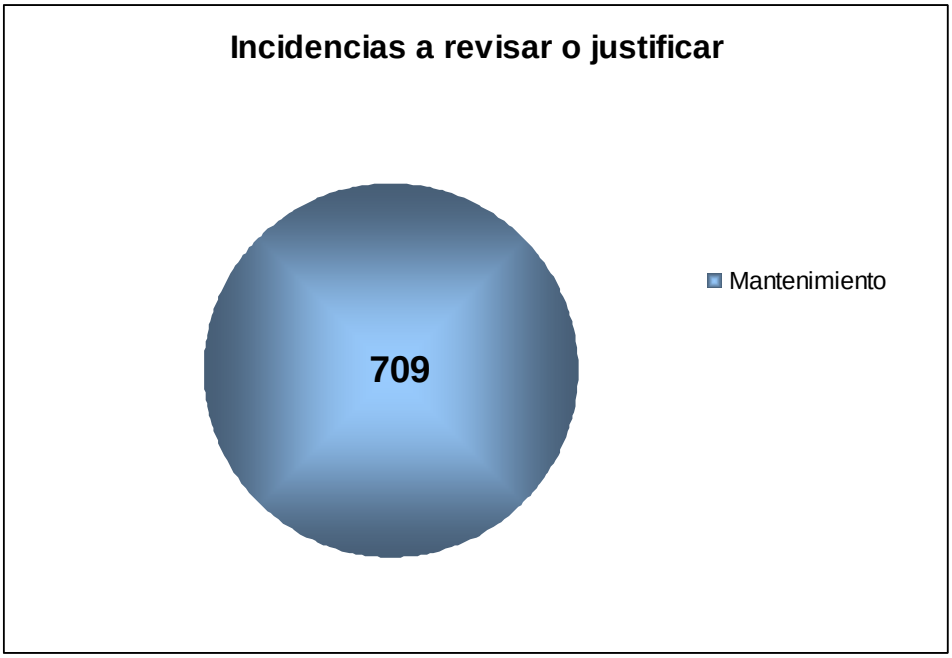


Figura 18 Incidencias a revisar o justificar.

Por severidad

En la Figura 19 se puede ver que las incidencias en su mayoría tienes una severidad media, pero no hay incidencias con severidad alta.

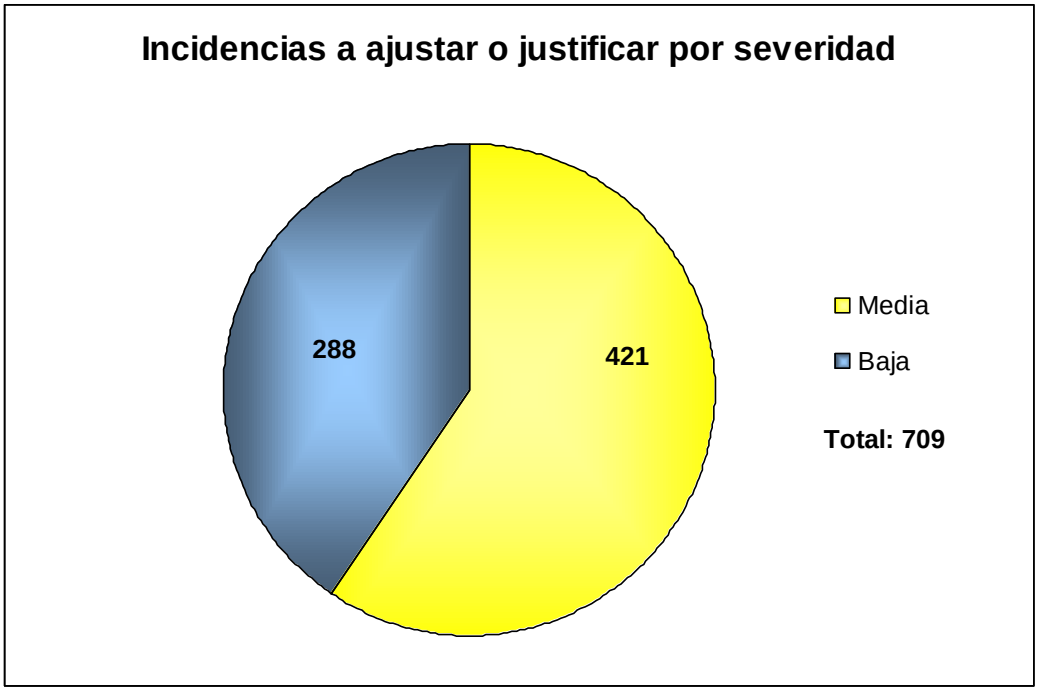


Figura 19 Incidencias a ajustar o justificar por severidad.

7.3.4 Resumen total de incidencias

Por factor de calidad

La Figura 20 muestra el total de incidencias por factor de calidad. En esta gráfica se puede observar como la gran mayoría de las incidencias afectan el mantenimiento.

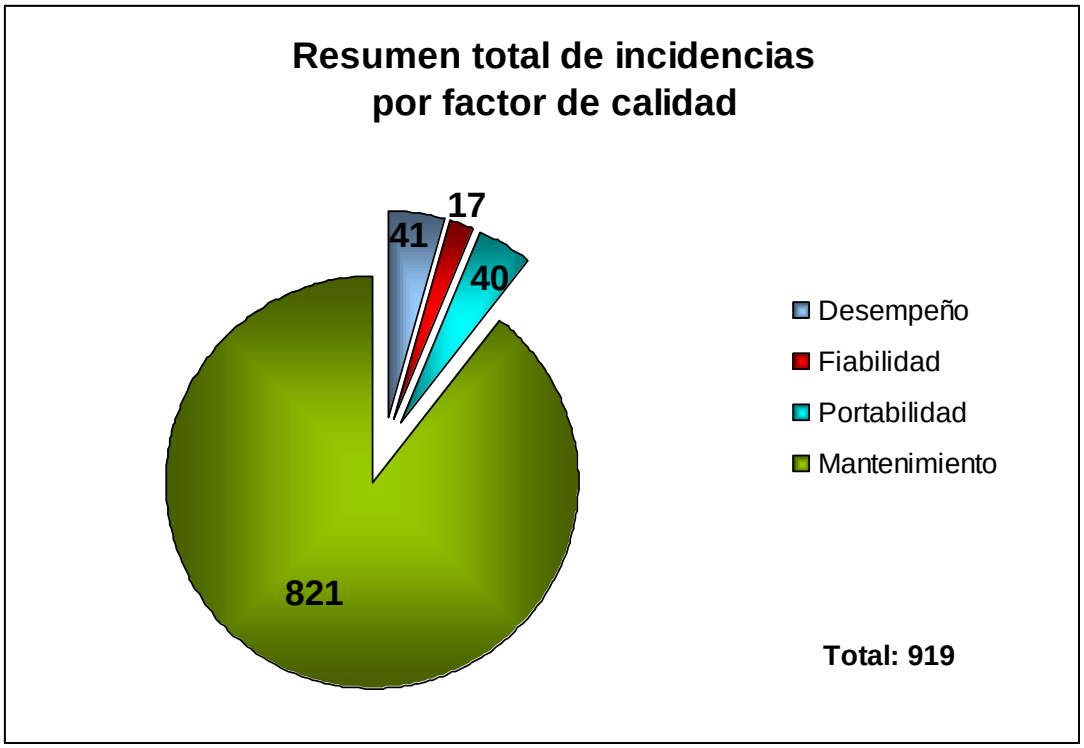


Figura 20 Resumen total de incidencias por factor de calidad.

Por Categoría

En la Figura 21 se muestran las incidencias clasificadas por categoría de incidencias según la herramienta CodePro.

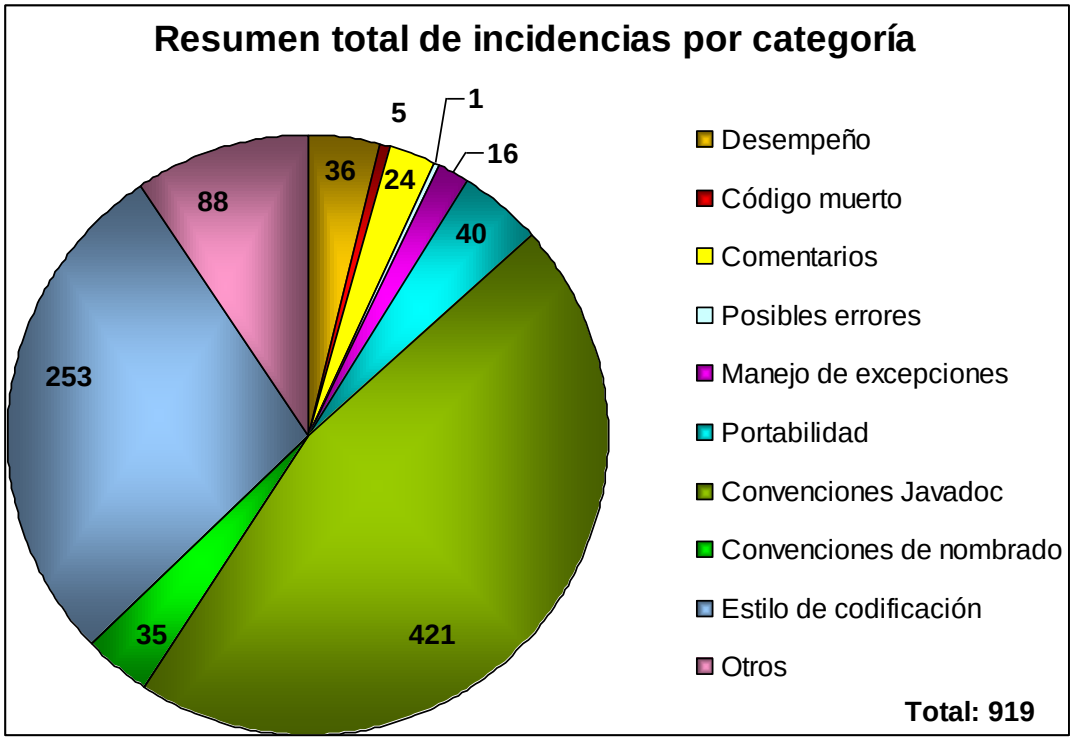
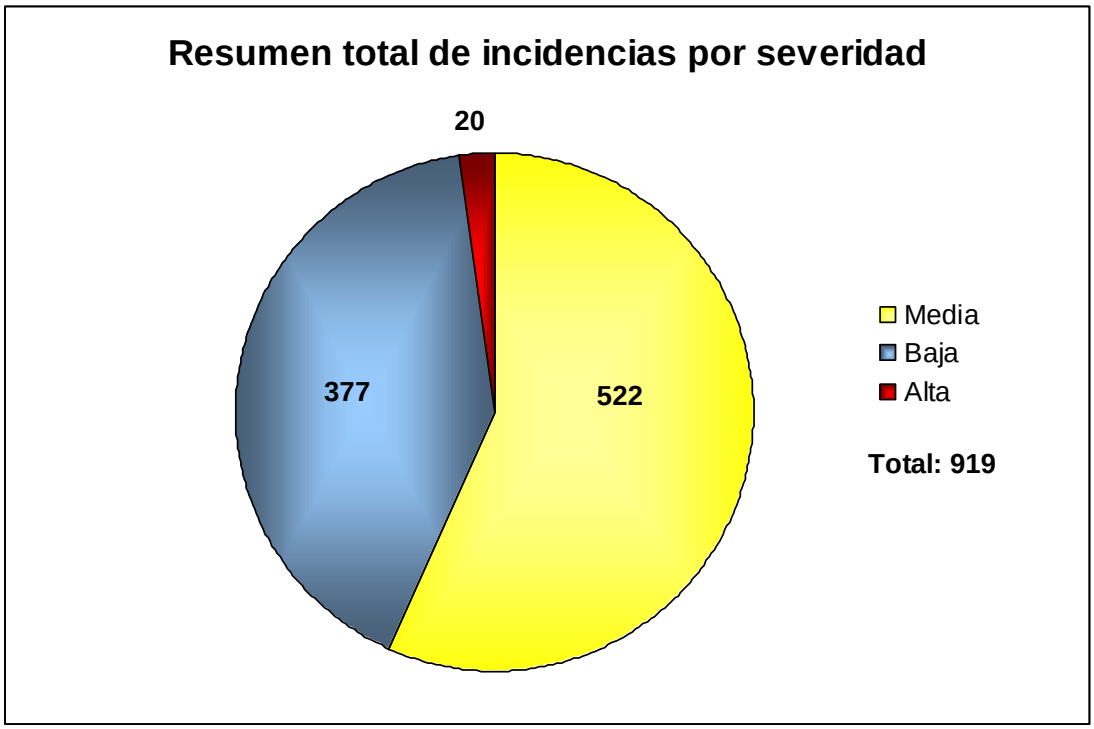


Figura 21 Resumen total de incidencias por categoría.

Por severidad

En la Figura 22 se ve claramente que las incidencias con severidad alta son muy pocas.



Gráfica 11. Resumen total de incidencias por severidad.

.

8 Conclusiones

Como resultado de la revisión y análisis del Módulo de Operación de Efectivo con base en las métricas en sus diferentes grupos, se pudo verificar que el sistema no presenta riesgos considerables en su estructura de código fuente. Las métricas de tamaño del sistema y complejidad dieron evidencia de que se trata de un sistema pequeño, acotado y que no se generarán altos costos de mantenimiento.

A pesar de esto, se recomendó al cliente corregir las incidencias registradas, ya que sí se encontraron puntos de mejora que si no se les da seguimiento, pueden hacer al sistema susceptible a fallos, como es el caso de las clases identificadas como complejas. Así también se sugirió hacer una revisión al soporte documental del código fuente, ya que los casos identificados pueden repercutir en el mantenimiento del sistema.

Asimismo, se retroalimentó al patrocinador del proyecto, informándole que las revisiones de calidad a sistemas de software otorgan un valor considerable al grupo de desarrollo, para la detección y corrección de errores, sobre todo cuando se realizan en etapas iniciales del desarrollo, ya que su mantenimiento es considerablemente menor en costo y esfuerzo, que cuando ya está construido el sistema.

Finalmente, cabe mencionar que la experiencia y aprendizaje obtenidos de la participación en este proyecto fue de gran valor, debido al apego a las áreas de conocimiento de ingeniería en computación, y que permitió reforzar los conocimientos adquiridos sobre ingeniería de software y administración de proyectos de tecnología.

9 Referencias

- Booch, et al,The Unified Modeling Language Reference Manual, Addison Wesley,1999
- Booch, et al, The Unified Modeling Language User Guide, Addison Wesley, 1998
- Cantor, Murray, Object Oriented Project Management with UML, Prentice-Hall, 2000.
- Goodman, Paul, Implementación práctica de Métricas de Software, McGraw-Hill, 1993.
- Hudli, Hoskins, Métricas de software para diseños orientados a objetos, IEEE International Conference on Computer Design, 1994.
- http://www.sei.cmu.edu/reports/88cm012.pdf
- Kan, Stephen H., Métricas y modelos en Ingeniería de Software de calidad, Addison Wesley, 2002.
- Marciniak, John J., ed. *Encyclopedia of Software Engineering,* pp. 131-165, New York, NY, John Wiley & Sons, 1994.
- McCabe, Thomas J. & Butler, Charles W., Design Complexity Measurement and Testing, *Communications of, the ACM 32,* pp. 1415-1425, 1989
- McCabe, Thomas J. & Watson, Arthur H., Software Complexity, *Crosstalk, Journal of Defense Software Engineering 7*, pp. 5-9, 1994.
- McConell, Steve, Code Complete 2nd Edition, Microsoft, 2004.
- Perry, William E., *A Structured Approach to Systems Testing,* Wellesley, MA: QED Information Sciences, 1988.
- Pressman, Roger S., Ingeniería de software, McGraw-Hill, 2006.
- Schmuller, Aprendiendo UML en 24 horas, Prentice Hall
- Unified Modeling Language Specification™ (UML®) ver. 1.4.2, www.omg.org
- Watson, Arthur H. & McCabe, Thomas J., Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, 1996.