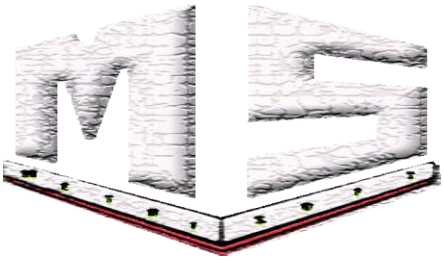
UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
Facultad 9

TÍTULO: Herramienta de software para automatizar la medición de productos de software en los proyectos productivos de la Facultad 9.



TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA

AUTORES: Lourdes Garcia Pérez
Dubiel Santana Monzón

TUTOR: Ing. Enrique Pérez Rodríguez

Ciudad de la Habana 28 de junio de 2010
"Año 52 de la Revolución."

Declaración de Autoría

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

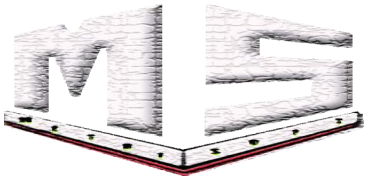
Lourdes Garcia Pérez

Dubiel Santana Monzón

Ing. Enrique Pérez Rodríguez

_____ _____

(Autores) (Tutor)



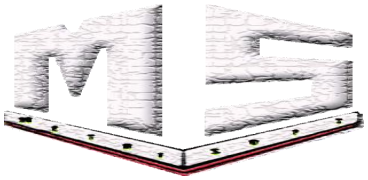
RESUMEN

Los métodos para producir *software* están en constante evolución proporcionando grandes demandas en ramas como la economía y la sociedad, surgiendo la necesidad de garantizar *software* de calidad. La Universidad de las Ciencias Informáticas (UCI), siendo el principal productor de *software* del país está vinculada al estudio y aplicación de las métricas, específicamente para dar respuesta a dicha necesidad, métricas sobre el producto. Desafortunadamente todavía no se es consciente del uso de las mismas en los procesos de medición, incluso dentro de la propia comunidad informática.

De ahí la importancia de querer lograr que se controlen y evalúen los diferentes productos de *software* permitiendo medir cuantitativamente cuánto se ha avanzado en un proyecto, en qué se debe mejorar, y proveer una mejor toma de decisiones por parte de los directivos del mismo, el presente trabajo centra su objetivo principal en la implementación de una herramienta de *software* que automatice el cálculo y la recopilación de los datos de las métricas sobre el producto generadas en los proyectos productivos de la Facultad 9. Para alcanzar dicho objetivo se realiza el proceso de desarrollo de la herramienta de *software* en su ciclo completo haciendo uso de las diferentes tecnologías y tendencias necesarias, el esbozo de la arquitectura del *software*, su diseño, la creación de la base de datos para el manejo de los datos, la implementación y por último el proceso de pruebas a la aplicación resultante.

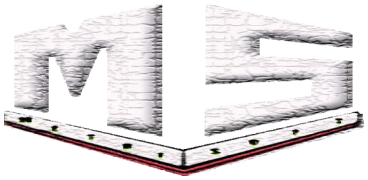
PALABRAS CLAVES

Automatización, Métricas, Medición, *Software*.



ÍNDICE

ÍNDICE



ÍNDICE

INTRODUCCIÓN

Introducción

Un *software*, casi nunca es perfecto. Para lograr la calidad del *software* se han dedicado muchos esfuerzos desde la década del setenta. El objetivo de cada proyecto es realizar productos de *software* con la mejor calidad dentro de lo posible, cumpliendo y mejor aún, superando las expectativas de los usuarios.

Una vez que los desarrolladores empiezan a generar el código, se preocupan por la calidad y sucede que esta es una actividad que debe desenvolverse durante todo el Proceso de Desarrollo de *Software*, desde etapas anteriores a esta. La preocupación por ofrecer productos que posean altos niveles de calidad, no es nueva. A lo largo de este siglo han surgido distintas interpretaciones de cómo brindar calidad. Actualmente pueden describirse algunas de las características que hacen comparable un producto de otro; se pueden considerar tamaños, formas, manejabilidad, colores, etc., características que son fáciles de comparar.

De la misma forma que existen medidas para estos atributos físicos, para el producto de *software* existen medidas que pueden hacerlo comparables: líneas de código, puntos de función. El principal propósito de un equipo de desarrolladores de *software* es producir siempre un *software* de la más alta calidad, que no es más que la "concordancia del *software* producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario" **(Pressman R. , 2002)** . Para ello se debe tener en cuenta que los productos de *software* son realizados por personas para personas; por lo que los desarrolladores deben contar con que son otras personas las que utilizarán sus productos, que pueden contener fallos constantes. La calidad no es un atributo exclusivo de los productos.

Comienza en Cuba una nueva era a partir del año 1996, dando los primeros pasos en impulsar el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC). La falta de un proceso de medición que propiciara evaluar la calidad del *software*, influyó en la realización de los productos, al surgir demoras en la entrega de los mismos al no cumplir con los requerimientos del cliente, iniciando así una crisis del *software* que empeoraba gradualmente debido a la limitación de recursos existentes en el país.

INTRODUCCIÓN

En la actualidad, el desarrollo de *software* aumenta considerablemente, presentando una tendencia de realizarse sin documentación y sin organización; de ahí la gran importancia de la necesidad de empezar a utilizar estándares y metodologías que hagan posible la uniformidad en el trabajo, logrando un mayor control en la calidad del producto. “Por esto se ve parte de la solución en la utilización de métricas durante todo el ciclo de desarrollo del software enfatizando en las primeras etapas a las que menos se le ha dedicado en lo que a ello respecta”. **(Alvarez, 2007)**

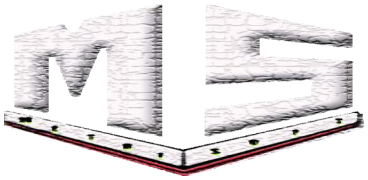
Las métricas son un buen medio para entender, controlar, monitorizar y predecir el desarrollo del *software*, la IEEE[1], define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Estas pueden ser utilizadas para la evaluación de las características del *software*. Según Pressman son: “Una serie de medidas o pasos que ayudan a definir con mayor exactitud el desarrollo y calidad de un producto.” **(Pressman R. S., 2002)**. Están basadas en técnicas de mediciones que muestran resultados cuantitativos de todo el proceso de desarrollo del *software*.

La información relacionada con las métricas de calidad tiene un auge cada vez mayor con el paso de los años, debido a esto, Martín dice que: “urge el disponer de repositorios genéricos y herramientas de catalogación que faciliten de un modo extensible y consensuado, el almacenamiento, consulta, explotación y reúso de toda la información relacionada a métricas que sirva de soporte a las actividades de aseguramiento de la calidad.” **(Martin, Bertoa, Valecillo, & Orsina, 2002)**. A partir de este momento empiezan a surgir paulatinamente herramientas que controlan, gestionan y calculan, estas métricas.

La Universidad de las Ciencias Informaticas (UCI), surgió con el propósito de crear *software* y su poco tiempo de existencia le trae como consecuencia contar con pocas herramientas en su campo productivo, herramientas que estén a cargo de la medición sistemática, así como la medición de productos de *software* en los proyectos productivos de la Facultad 9.

Entonces, la presente propuesta surge debido a la necesidad del Grupo de Calidad de la Facultad 9 de la UCI de hacer una herramienta de software para automatizar las métricas del producto de *software*,

[1] **Instituto de Ingenieros Electricistas y Electrónicos.**



respondiendo a las perspectivas de dicha facultad ante la dificultad en el entorno de desarrollo del producto de software en los proyectos productivos, los que se ven afectados por la no utilización de las métricas para evaluar la calidad del producto.

No se realizan las mediciones de forma periódica, no existe un procedimiento para la recopilación de los datos de las mismas y por consiguiente, no se tiene un registro de todas las mediciones que se efectúan, todo esto introduce el **problema a resolver**: la no definición de un proceso de medición y por tanto la no implementación del mismo en los proyectos productivos de la Facultad 9, trae como consecuencia que se le dificulte la toma de decisiones a los directivos de dicha facultad.

El **objeto de estudio** lo comprende la medición de productos de *software* en los proyectos productivos de la Facultad 9.

Procediéndole el **campo de acción** la automatización del proceso de medición de productos de *software* en los proyectos productivos de la Facultad 9.

El **objetivo general** que se plantea es desarrollar una herramienta de *software* que permita automatizar el proceso de medición de productos de *software* en los proyectos productivos de la Facultad 9.

Con la siguiente **idea a defender**: el desarrollo de una herramienta de *software* que automatice el proceso de medición de productos de *software* en los proyectos productivos de la Facultad 9 permitirá facilitar los procesos de medición para efectuar la toma de decisiones por parte de los directivos de dicha facultad.

Para dar solución a la situación problemática y cumplir con el objetivo planteado, se trazaron las siguientes **Tareas de la investigación**:

1. Analizar de forma exhaustiva las tendencias actuales del empleo del proceso de análisis de productos de software en la Universidad de las Ciencias Informáticas.
2. Seleccionar las métricas del producto de *software* que se van a implementar en el sistema.
3. Seleccionar las herramientas y tecnologías a utilizar para el desarrollo del sistema.

4. Modelar una herramienta de *software* que permita monitorizar el proceso de medición de productos de *software* en los proyectos productivos de la Facultad 9.
5. Diseñar una Base de Datos que soporte el manejo de datos de las funcionalidades del sistema.
6. Implementar dicha herramienta de *software* que permita automatizar el proceso de medición de productos de *software* en los proyectos productivos de la Facultad 9.
7. Probar la herramienta implementada.

Para la realización de las tareas mencionadas anteriormente se emplearon **métodos científicos**. Los mismos se dividen en teóricos y empíricos.

Los **métodos teóricos** posibilitan las condiciones para buscar más que las características triviales de la realidad, permiten explicar los hechos y profundizar en las principales relaciones y cualidades de los fenómenos, hechos y procesos. En este grupo se utilizan:

- El método Análisis Histórico-Lógico, está vinculado a la evolución del fenómeno en cuestión, utilizado para identificar la existencia de una herramienta que de solución al fenómeno de investigación. Este método permite desarrollar el estudio del arte previo al desarrollo de la investigación.

- El Analítico-Sintético que se utiliza para entender y plantear a partir de fuentes bibliográficas seguras la importancia y el uso de las métricas como estándares en todo el proceso de medición de la calidad de procesos de desarrollo del *software*.

- El método de Modelación permite prever la respuesta de un proceso al aplicarle algunas diferencias en sus parámetros sin tener que necesariamente ejecutar el proceso en la realidad. A partir de este método se va a brindar información de lo que se está estudiando.

Los **métodos empíricos** revelan, describen y explican las características y relaciones esenciales del objeto basando su contenido en la experiencia. Dentro de este otro grupo se utilizan los métodos:

- Entrevistas, estas son aplicadas a los líderes de los proyectos productivos de la Facultad 9, para adquirir la mayor cantidad de información posible sobre los procedimientos de medición y análisis de productos de *software* en sus respectivos proyectos productivos, además de la recopilación de estos datos.

- Encuesta que es aplicada a los jefes de Departamentos de Centro de la Facultad 9 con el objetivo de conocer las prácticas que se realizan en los proyectos que atienden para lograr un mejor proceso de medición y análisis del producto de *software*. Además de investigar los modelos de calidad, estándares o normas que emplean para sustentar lo anterior.

Haciendo posible que la investigación esté basada en elementos creíbles, se enfoca el trabajo en la técnica de muestreo No probabilística: Muestreo Intencional, esta es la más aconsejada a utilizar porque permite agrupar a los individuos que mayor cantidad de información van a proporcionar.

En el transcurso de la aplicación de las encuestas se utilizó una:

Población: Los Jefes de Departamentos de Señales Digitales y GeoInformática de la Facultad 9. (Total: 2)

Muestra: Dos Jefes de Departamento Centro de la Facultad 9.

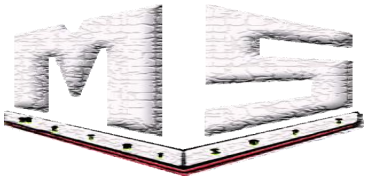
Unidad de estudio: Un Jefe de Departamento Centro de la Facultad 9.

De la misma forma se ejecutó el proceso de realización de entrevistas con la selección de una:

Población: Líderes de los proyectos productivos de la Facultad 9. (Total: 8)

Muestra: Cinco líderes de proyectos productivos de la Facultad 9.

Unidad de estudio: Un líder de proyecto de la Facultad 9.



CAPITULO 1

CAPÍTULO 1: Fundamentación Teórica.

1.1 Introducción

"El producto se mide para intentar aumentar su calidad". **(Pérez, 2007)**. Cuando se empieza a planificar un proyecto se tienen que obtener estimaciones del esfuerzo humano y costo necesitados a través de las mediciones de *software* que se emplean para reunir la mayoría de los datos cualitativos acerca del producto de *software* y sus procesos.

En la actualidad existen muchos desafíos técnicos generados por el constante desarrollo existente en las TIC[2], y las métricas son un medio para ayudar a entender el proceso que se utiliza para desarrollar un producto, además del producto en sí. Tal parece que la necesidad de la medición es algo incuestionable, sin lugar a dudas, es lo que permite cuantificar y gestionar de la forma más efectiva.

Todo esto trae consigo que surjan las siguientes dudas: ¿cuáles serían las métricas apropiadas para el producto?, ¿cómo utilizar los datos que se recopilen? Preguntas que siempre surgen cuando no se ha medido un proyecto con anterioridad.

Este capítulo aborda los conceptos principales asociados al problema planteado, seleccionando algunas de las métricas del producto que se van a utilizar en la implementación de la herramienta a desarrollar.

1.2 ¿Qué es software?

Hasta principios de la década de los sesenta fue que se acuñó el término de "*software*". Las primeras computadoras solo ejecutaban programas y no existía este concepto. Surge entonces, para poder diferenciar los sistemas operativos de los programas de tipo matemático para las que fueron construidas las primeras computadoras.

[2] **Tecnologías de la Información y las Comunicaciones**

Por lo que el *Software* es ese soporte lógico que permite que la computadora pueda realizar tareas inteligentes, dirigiendo al hardware con instrucciones y datos a través de diferentes tipos de programas.

El software posibilita la relación entre el ser humano y la máquina y también de las máquinas entre sí. Sin ese conjunto de instrucciones programadas, los ordenadores serían objetos indiferentes.

1.2.1 Antecedentes al concepto de calidad

La búsqueda y la dedicación por la perfección por parte del hombre, siempre ha sido de forma constante, por lo que la necesidad de asumir responsabilidades sobre el trabajo que se efectúe y el interés por este bien hecho, derivó paulatinamente el concepto de calidad.

Uno de los ejemplos más tempranos que se tiene es la construcción de las famosas pirámides de Egipto; el acercamiento a la perfección de estas edificaciones se consiguió gracias a los métodos de inspección empleados durante su construcción.

Luego en la edad media surgió en Europa el sistema de organización en gremios, estos imponían los precios y especificaciones de los diferentes productos que proveían a la sociedad; el producto entre más calidad tenía, pues más prestigio le daba al artesano.

Comienza a desaparecer el artesanado cuando llega la revolución industrial, es entonces cuando aparecen los trabajadores de empresas, y posterior a estos, los empleados dedicados a la inspección, aunque al principio se le prestaba más atención a la forma de realizar el trabajo, o sea, a los procesos, que a la calidad de los productos.

Al Dr. Walter A. Shewhart se le conoce como el padre del control estadístico de la calidad, debido a que finalmente en los años 30 del siglo XX se inició el control de calidad moderno o control de calidad estadístico ideado por él. Posterior a esto ya en Japón se introdujo la idea de que la calidad de un producto o servicio residía en el grado de mentalización de todo el personal de la organización.

Hubo una transición en las actividades de control de la calidad, debido a seminarios que se impartieron a los mandos altos donde se les explicaba las funciones que les correspondía a cada uno en la promoción de la misma; fue creándose un ambiente de Control de la calidad hasta que se concibió como el de hoy en día, estableciendo un total control sobre la calidad.

1.2.2 ¿Qué es calidad?

En el epígrafe anterior se enunciaban algunos de los que fueron antecedentes de lo que es hoy la calidad. Evolucionando desde Control de la Calidad hasta Calidad total.

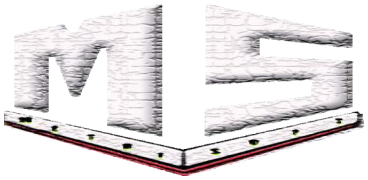
Feigenbaum, a quien se le debe el término de calidad total, la define “para quien el objetivo es satisfacer al cliente**,** y la forma de lograrlo es la mejora continua de la calidad”. **(Feingenbaum, 1961)**

Según el diccionario, calidad se puede definir como "una característica o atributo de una cosa". Así de esta forma se puede decir que la calidad de los productos podría medirse como una comparación de sus características y atributos. De esta manera este concepto se puede aplicar a cualquier producto. El brindar calidad es una actividad esencial para un negocio que produce productos que serán utilizados por otras personas.

La calidad es un concepto que ha ido variando con los años y existe una gran variedad de formas de concebirla. Ha pasado a ser definida por grandes estudiosos del tema:

Kaoru Ishikawa, define la calidad como: “Desarrollar, diseñar, manufacturar y mantener un producto de calidad que sea el más económico, el más útil y siempre satisfactorio para el consumidor”. **(Ishikawa, 1998)**

Pressman la define como “concordancia del software con los requisitos explícitamente establecidos, con los estándares de desarrollo expresamente fijados y con los requisitos implícitos, no establecidos formalmente que desea el usuario”. **(Pressman, 1993)**



Estos autores han influenciado de forma directa y excepcional en el desarrollo y evolución del concepto actual de calidad. De ahí que se afirma que la calidad es ese proceso imprescindible para la evaluación y el control de las especificaciones requeridas con las que debe cumplir el producto final antes de ser entregado a su cliente y afecta a todos los profesionales y procesos implicados.

1.2.3 Calidad en los productos de software

La calidad en los productos de *software* es una necesidad para todas las empresas. El interés por la calidad aumenta continuamente en la medida en la que los clientes se vuelven más selectivos y empiezan a rechazar los productos que no satisfacen sus necesidades. Por lo que a la hora de definir la calidad de *software* se tiene que tener en cuenta tanto la calidad del producto de *software* como la calidad del proceso de desarrollo del *software* porque la calidad del producto va a estar en función de la calidad del proceso de desarrollo. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto.

La calidad del producto de *software* se diferencia de la calidad de otros productos de fabricación industrial, ya que el *software* tiene ciertas características especiales como que el *software* es un producto mental, o sea, es algo abstracto y por tanto, su calidad también lo es; el mismo se desarrolla, no se fabrica, implicando que el coste sea precisamente en el proceso de diseño; este no se deteriora con el tiempo y todos los problemas que surjan, estaban allí desde el principio; mantener el *software* es mucho más complejo que mantener el *hardware* debido a que al deteriorarse un componente *hardware* sencillamente se sustituye por una pieza de repuesto, a diferencia del *software* que cada fallo indica un error en el proceso en el cual se tradujo el diseño en código máquina ejecutable; se pueden realizar cambios de forma muy fácil sobre un producto de *software*, cambios que pueden llegar a ser muy difíciles de controlar si no se lleva el debido seguimiento con ellos.

Existen infinitas definiciones de calidad de *software* tales como:

Calidad de *software* “es la medida en que las propiedades de un bien o servicio cumplen con los requisitos establecidos en la norma o especificaciones técnicas, así como con las exigencias del usuario de dicho bien o servicio en cuanto a su funcionalidad, durabilidad y costo”. **(SEI, 2002)**

CAPITULO 1

La norma ISO[3] 8402 define que “la calidad es la suma de todos aquellos aspectos o características de un producto o servicio que influyen en su capacidad para satisfacer las necesidades, expresadas o implícitas”. **(Antonio, 2006)**

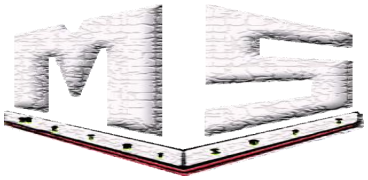
La norma IEEE 729-83 la define como el “grado con el cual el cliente o usuario percibe que el software satisface sus expectativas” **(Antonio, 2006)**

Teniendo en cuenta los conceptos anteriores se determina que el grado de complicidad en satisfacer las necesidades y expectativas del usuario, haciendo posible cumplir con las exigencias del mismo sería lo que define la calidad de software.

La calidad es un conjunto de características y propiedades que representan una ventaja estratégica permitiendo actuar sobre las tareas menos eficientes, mejorándolas. Para ello el lector se puede apoyar en los factores de calidad definidos por McCall:

Figura 1: Factores de Calidad del Software McCall. (Ruilova Rojas, 2008)

[3] **Organización Internacional para la Estandarización**



CAPITULO 1

La calidad del *software* es medible y varía de un sistema a otro o de un programa a otro, puede medirse después de elaborado el producto, pero esto puede resultar muy costoso, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida del *software*, es decir, debe correr paralela desde la planificación del producto hasta la fase de producción del mismo.

1.2.4 ¿Qué es medición?

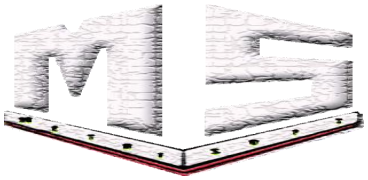
A la hora de realizar un *software*, el cliente espera que este - al igual que otro producto que afecta su vida cotidiana- cumpla con ciertos requisitos de calidad. Se hace necesario un control de todo el proceso de desarrollo teniendo en cuenta la tecnología de ingeniería de software apropiada, con el fin de que el proceso sea preciso, predecible y repetible. Un factor clave para alcanzar estos objetivos es la medición.

¿Entonces qué es medición? **:** Establece la terminología relacionada con el acto de medir el *software*. Una medición (que es una acción) es un conjunto de operaciones cuyo objetivo es determinar el valor de un resultado de medición para un dado atributo de una entidad, utilizando una forma de medir. “Los resultados de la medición se obtienen a través de la acción de llevar a cabo una medición.” **(Ferreira, Garcia, Ruiz, & Bertoa, 2006)**

La medición del producto realizada durante las primeras etapas del proceso de *software* proporciona a los ingenieros un mecanismo consistente y objetivo para evaluar la calidad.

Hay varias razones para medir un producto a decir de Otoniel:

1. Para indicar la calidad del producto.
2. Para evaluar la productividad de la gente que desarrolla el producto.
3. Para evaluar los beneficios en términos de productividad y de calidad, derivados del uso de nuevos métodos y herramientas de la ingeniería de *software*.
4. Para establecer una línea de base para la estimación.
5. Para ayudar a justificar el uso de nuevas herramientas o de formación adicional. **(Pérez, 2007)**



La medición del *software* juega un papel muy importante en la Ingeniería del *Software*. En la actualidad las métricas del *software* han demostrado ser muy eficaces en la construcción de sistemas de alta calidad para proyectos grandes de bases de datos; en la evaluación y garantía de calidad de sistemas; en la comprensión y mejora de los proyectos de desarrollo y mantenimiento del *software*, etc.

Además, las métricas de *software* son herramientas importantes que ayudan en la evaluación y en la institucionalización de la Mejora del Proceso Software en organizaciones que lo desarrollan.

De hecho, la medición del *software* es la pieza clave de iniciativas como SW-CMM (*Capability Maturity Model for Software* – Modelo de Madurez de Capacidad para Software), ISO/IEC 15504 (SPICE, *Software Process Improvement and Capability Determination* – Mejora del Proceso Software y Determinación de Capacidad) y CMMI (*Capability Maturity Model Integration* – Integración del Modelo de Madurez de Capacidad). El estándar ISO/IEC 90003:2004 también destaca la importancia de la medición en la gestión y garantía de la calidad.

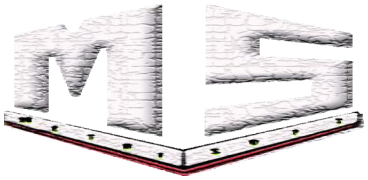
1.2.5 CMM-CMMI: Aseguramiento de la calidad

El *software* puede ser considerado como producto o como servicio. Lo que ha motivado a escala mundial, generar un conjunto de modelos para estimar su calidad, los cuales responden a las necesidades de garantizar productos de calidad.

Los CMM[4] se concentran en la mejora de los procesos de una organización. Contienen los elementos esenciales de eficacia de los procesos en una o más disciplinas y describen un camino de mejora evolutivo que permite pasar desde procesos inmaduros a procesos disciplinados y maduros de mejor calidad y más eficaces. En la actualidad CMM está sustituido por CMMI[5].

CMMI es un modelo de referencia que cubre las actividades del desarrollo y del mantenimiento aplicadas tanto a los productos como a los servicios.

[4] Modelo de Madurez de Capacidad
[5] Integración del Modelo de Madurez de Capacidad



CAPITULO 1

"CMMI (*Capability Maturity Model Integration*) es un modelo de madurez de mejora de los procesos para el desarrollo de productos y de servicios. Consiste en las mejores prácticas que tratan las actividades de desarrollo y de mantenimiento que cubren el ciclo de vida del producto, desde la concepción a la entrega y el mantenimiento". **(Chrissis, Konrad, & Shrum, 2009)**

El propósito de CMMI para desarrollo es ayudar a las organizaciones a mejorar sus procesos de desarrollo y de mantenimiento, tanto para los productos como para los servicios. Los modelos CMMI describen las que han sido consideradas como mejores prácticas que las organizaciones han encontrado productivas y útiles para conseguir sus objetivos de negocio.

El modelo CMMI se representa de forma continua y por etapas.

La representación continua permite seleccionar un área de proceso y mejorar los procesos relacionados, utilizando seis niveles de capacidad de CMMI que hacen referencias a las mejoras concernientes a un área de proceso individual.

- Nivel 0: Incompleto.
- Nivel 1: Realizado.
- Nivel 2: Administrado.
- Nivel 3: Definido.
- Nivel 4: Administrado cuantitativamente.
- Nivel 5: En optimización.

La representación por etapas utiliza conjuntos de áreas ya precisadas para definir un camino de mejora para una organización, esta representación utiliza cinco niveles de madurez de CMMI, viabilizando cada uno de ellos un grupo de áreas de proceso caracterizando diferentes comportamientos organizativos:

- Nivel 1: Inicial.
- Nivel 2: Administrado.
- Nivel 3: Definido.

- Nivel 4: Administrado cuantitativamente.
- Nivel 5: Optimizado.

Para establecer la capacidad de la organización o madurez organizacional, CMMI define 22 áreas de procesos, en cada una de ellas se agrupan un conjunto de prácticas relacionas que cuando se implementan de forma paralela, satisfacen un grupo de objetivos considerados fundamentales para la mejora de la tarea.

1.2.5.1 Área de proceso de CMMI. Medición y análisis (MA).

El propósito de esta área de proceso de CMMI es desarrollar y sostener una capacidad de medición para apoyar las necesidades de información de la administración. Donde los objetivos y las actividades de medición estén alineadas con los objetivos y las actividades de información identificadas. Proporcionando además, los resultados de estas mediciones, que están dirigidos precisamente a estos objetivos y necesidades de información.

Las necesidades informacionales se crean a partir de las características específicas de cada organización. Están generalmente concentradas en indicadores como: “el costo, la calidad, cronogramas, la satisfacción del cliente o la generación de nuevos negocios”. **(Kulpa & Kent, 2008)**

Para reconocer las necesidades que presenta la organización se deben tener en cuenta algunos documentos que pueden proveer algunas pistas como los que se nombran a continuación:

- “Planes (por ejemplo: planes de proyectos, planes estratégicos, planes de negocio y planes de mejora de procesos).
- Resultado del monitoreo del avance del proyecto.
- Administración de objetivos.
- Requerimientos formales u obligaciones contractuales.
- Administración recurrente o problemas técnicos.” **(Kulpa & Kent, 2008)**

La Universidad de las Ciencias Informáticas se encuentra en un proceso de mejora de desarrollo con el que se desea lograr alcanzar el nivel 2 de madurez de CMMI, y de esta forma poder llegar a implementar con eficiencia las áreas de procesos definidas en este nivel, siendo de mucha utilidad para este trabajo la utilización específicamente del área de proceso de Medición y Análisis (MA).

1.2.6 Métricas de software

"Debemos recordar que otras disciplinas científicas ya han aplicado los conceptos básicos de la medición. En Ingeniería del *Software* no hay que reinventar demasiado, simplemente aplicar y adaptar la teoría ya existente a las métricas del software." **(Dolado, 2000)**

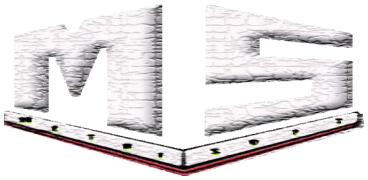
"Las métricas del *software* permiten medir de forma cuantitativa la calidad de sus atributos internos del producto, esto permite al ingeniero evaluar la calidad antes de su construcción." **(Ruilova Rojas, 2008)**

El IEEE *Standard Glosary of Software Engeneering Term*" define las métricas como "una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado." **(Davis, 1993)**

No se debe olvidar que las métricas surgen como una herramienta para poder satisfacer las necesidades de información por parte de los usuarios, de una forma objetiva y cuantificable. Las métricas de *software* proveen toda la información necesaria para la toma de decisiones técnicas por parte de los directivos del proyecto en cuestión.

El IEEE define métrica como: "Una función que toma como entrada cierta información del software que se está midiendo, y que devuelve como salida un valor numérico sencillo, el cual es interpretada, como el grado en que el producto de software posee un atributo dado que afecta a su calidad." **(Vega Lebrún, Rivera Prieto, & Garcia Santillán, 2008)**

Las métricas se aplican tanto a los procesos (métricas de control): por ejemplo, tiempo y esfuerzo medios necesarios para corregir un error; como a los productos (métricas de predicción): complejidad ciclomática de un módulo, número de métodos y atributos asociados con los objetos de un diseño.



Permiten tomar decisiones por parte de los directivos administrativos de la organización donde se apliquen dichas métricas.

Estas se clasifican de la siguiente forma:

- **“Métricas técnicas:** Miden la estructura del sistema, el cómo está hecho, es decir, están centradas en las características del software más que en su proceso de desarrollo.
- **Métricas de calidad:** Proporcionan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente.
- **Métricas de productividad:** Se centran en el rendimiento del proceso de la ingeniería del software, es decir qué tan productivo va a ser el software que se va a diseñar.
- **Métricas orientadas al tamaño:** Es para saber en qué tiempo se va a terminar el software y cuantas personas se van a necesitar, son medidas directas al software y al proceso por el cual se desarrolla.
- **Métricas orientadas a la función:** Son medidas indirectas del software y del proceso por el cual se desarrolla, se centran en la funcionalidad o utilidad del programa.
- **Métricas orientadas a la persona:** Proporcionan medidas e información sobre la forma en que las personas desarrollan el software, son las medidas que se van a hacer del personal que hará el sistema.” **(Pressman R. S., 2002)**

CAPITULO 1

El Centro para la Excelencia en el Desarrollo de Proyectos Tecnológicos, conocido en la Universidad de las Ciencias Informáticas como CALISOFT, clasifica las métricas de la siguiente forma:

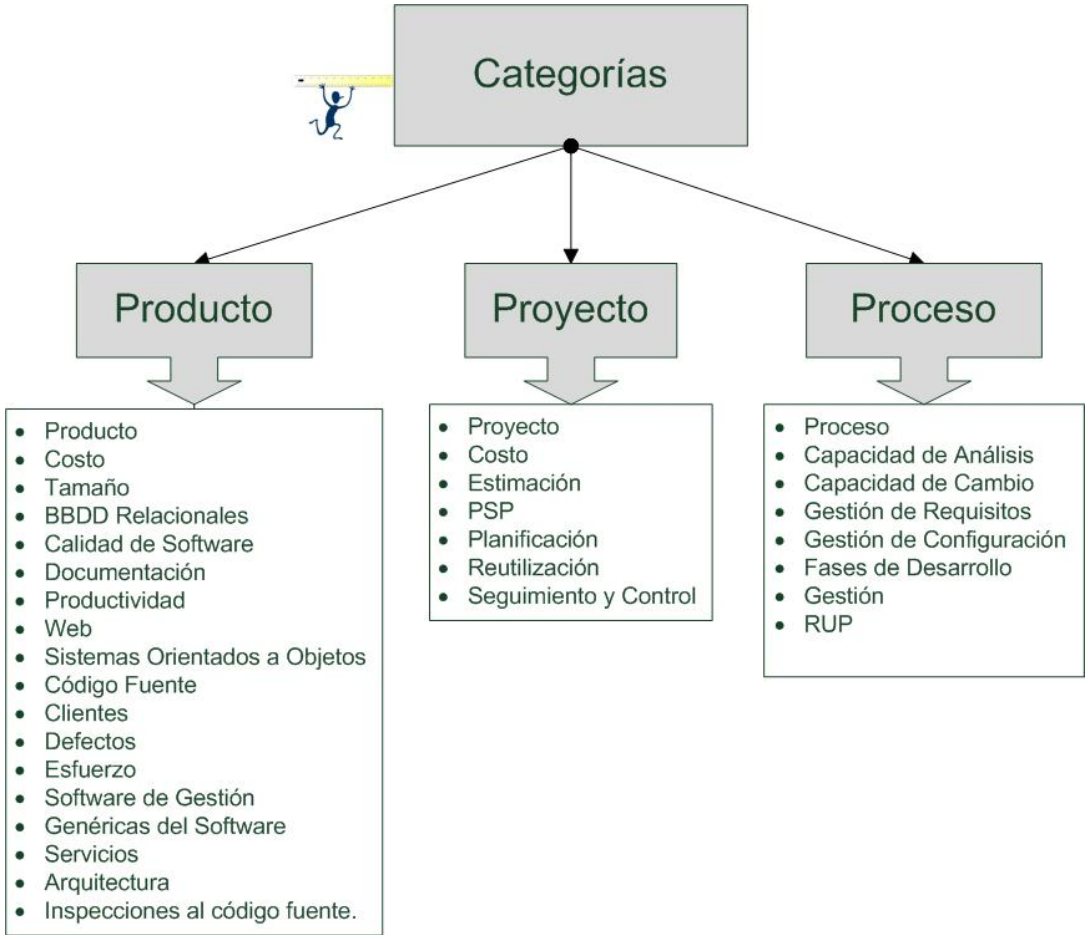
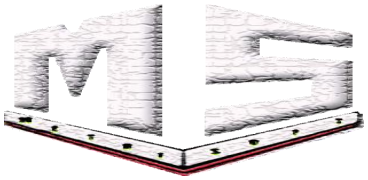


Figura 2: Diagrama para la clasificación de las métricas. (Rodríguez Verdecia & Lugo García, 2009)

Existe una gran gama de métricas para el proceso de desarrollo de *software*, pero no todas conforman un soporte práctico para el desarrollador. Incluso pueden llegar a causar confusión por la complejidad de sus mediciones. Debido a ello se considera que “las métricas deben ser: (1) simples, (2) objetivas, (3) fáciles de coleccionar, (4) fáciles de interpretar y difíciles de malinterpretar”. **(RUP, 2003)**



CAPITULO 1

1.3 Argumentación del Objeto de Estudio: Métricas del producto en el proceso de desarrollo de software

Cuando se hace referencia a las métricas del producto se refieren a las características del propio *software*, las relaciones entre características del *software* pueden variar dependiendo de diversos factores (proceso, tecnología de desarrollo, tipo de sistema).

Se argumenta que éstas deben ser enunciadas y utilizadas para administrar el proceso de desarrollo, y debe ser conforme al producto de *software* particular. El proveedor de productos de *software* debe de recopilar y actuar sobre las medidas cuantitativas de la calidad de esos productos de *software*.

Estas medidas deben ser utilizadas para los propósitos siguientes: recopilar información y reportar valores de métricas sobre bases regulares; identificar el actual nivel de desempeño por cada métrica; establecer metas de mejoras específicas en términos de las mismas métricas.

Panorama de las métricas del producto:

“Métricas para el modelo de análisis

- Funcionalidad entregada: medida indirecta
- Tamaño del sistema
- Calidad de la especificación

Métricas para el modelo de diseño

- Métricas arquitectónicas
- Métricas a nivel de componente
- Métricas del diseño de la interfaz
- Métricas especializadas del diseño orientado a objetos

Métricas para el código fuente

- Métricas de *Halstead*

- Métricas de complejidad
- Métricas de tamaño

Métricas para pruebas

- Métricas de cobertura de instrucciones y ramas
- Métricas relacionadas con los defectos
- Efectividad de las pruebas" **(Ruilova Rojas, 2008)**

En muchos casos las métricas de un modelo se utilizan en actividades posteriores de la Ingeniería del *Software*. Las métricas sobre el producto están orientadas a estimar las características del mismo antes de su desarrollo. Estas estimaciones se basan en el conocimiento que los desarrolladores adquieren a partir de datos obtenidos de proyectos anteriores.

1.3.1 Selección de las Métricas del Producto a utilizar en la propuesta

Las métricas que a continuación fueron seleccionadas para implementar la herramienta a desarrollar se hace, basándose en el hecho de que las mismas, ayudan a entender el proceso técnico que se utiliza para desarrollar un producto, como el propio producto. Además de ser las más proclives a utilizar en los proyectos productivos de la Facultad 9, debido a que son mediciones que se pueden evaluar de forma constante, incluso por los propios desarrolladores cuando la aplicación de la propuesta solución, esté terminada.

No.	Métricas del producto	Selección	
		SI	NO
1	Producto		X
2	Costo	X	
3	Tamaño		X
4	BBDD Relacionales		X
5	Calidad de Software		X
6	Documentación		X

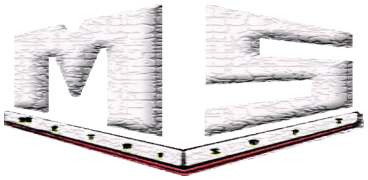
7	Productividad	X	
8	Web		X
9	Sistemas Orientados a Objetos	X	
10	Código fuente	X	
11	Clientes		X
12	Defectos		X
13	Esfuerzo		X
14	Software de Gestión	X	
15	Genéricas del Software		X
16	Servicios	X	
17	Arquitectura	X	
18	Inspecciones al Código fuente	X	

Tabla 1: Muestra de las métricas a implementar en la propuesta solución.

1.4 Descripción actual del dominio del problema

La situación actual del empleo de las métricas de calidad en la UCI, está basada en una estrategia introducida por la Dirección de Calidad de *Software* de la UCI garantizando el crecimiento continuo de una producción de *software* con calidad en la organización; a través de la definición de procesos siguiendo las especificaciones de metodologías, estándares y modelos de desarrollo de *software*, brindando asesorías, entrenamiento, métodos de medición y servicios de verificación-validación a las diferentes entidades.

Sin embargo, de acuerdo a los datos recogidos en las encuestas y entrevistas realizadas a líderes de proyectos de la Facultad 9, todavía existen problemas en las entregas de los productos de *software* con la calidad requerida. Han existido cambios recientes en la reestructuración de los departamentos de centro, o sea, en la ubicación de los proyectos dentro de áreas productivas.



CAPITULO 1

Proyectos productivos que en su mayoría cuentan con un Plan de Aseguramiento de la Calidad, pero que no lo hacen efectivo por no tener un personal encargado del rol de la medición específicamente, dificultando el progreso de los diferentes procesos debido a la inexistencia de registros históricos que permitan medir, evaluar y controlar el trabajo realizado.

Proyectos de *software* que a su vez no son correspondientes con el propósito de cumplir con solucionar problemas que surgen como pueden ser las correcciones, toma de decisiones, exceso de gasto, coste de mantenimiento, etc., olvidando que a partir de los procesos de medición se permitirá proporcionar requerimientos verificables en términos medibles; posibilitar evidencia cuantificable para tomar decisiones; se pueden identificar problemas de forma previa y así facilitar la visibilidad de todo el proceso de desarrollo; así como establecer predicciones de coste, y valorar efectos de calidad en los productos.

Es por ello que se requiere el monitoreo constante de las métricas sobre el producto que ayudarían a evaluar los procesos de desarrollo en los proyectos productivos de *software* en la Facultad 9, estando fundamentadas en parámetros objetivos y aportando una mejora organizada en los procedimientos actuales de trabajo.

1.5 Análisis de posibles soluciones existentes

Es de vital importancia conocer algunas de las posibles soluciones existentes tanto a nivel nacional como internacional que participen en el Control y Gestión de métricas para el desarrollo del *software*. En la UCI se han dado soluciones a aspectos que se relacionan con la actividad que se investiga, por lo que al apoyarse en estos aspectos se incrementa el soporte de credibilidad y validez de la solución que se quiere plantear.

1.5.1 Sistema de Gestión y Control de Métricas

En la Facultad 8 de la UCI se desarrolló una herramienta de *software* para el control y gestión de las métricas del desarrollo de *software*, esta no es utilizada en los proyectos productivos de la Facultad 9, además de no presentar los cálculos de las métricas del producto.

1.5.2 Modelo de Evaluación del Proceso de Desarrollo del Software Educativo (MEPDSE)

Modelo de Evaluación del Proceso de Desarrollo del *Software* Educativo que permite gestionar la calidad del proceso de desarrollo del *software* educativo, pero que necesita métricas para ser aplicable, pues en estos momentos depende del conocimiento de los evaluadores. (Oliva, 2008). Se realizó una propuesta de métricas que soportan este modelo pero su cálculo y gestión son completamente manuables. (Ver más detalles en trabajo de diploma: "Métricas para la evaluación del proceso de desarrollo de Software Educativo"). **(Nadereau, 2009)**

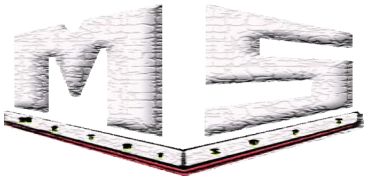
1.5.3 Calculador de Métricas de Calidad (CMC)

"Herramienta desarrollada en la Facultad 9 con el propósito de permitir medir cuantitativamente los atributos (efectividad, funcionalidad, fiabilidad, tiempo de respuesta, seguridad, confiabilidad, calidad, eficacia, mantenibilidad, eficiencia, usabilidad, integridad, corrección, etc.) de cada proyecto." **(Nadereau, 2009)**. Se realizó una propuesta de métricas de calidad en este trabajo, por lo que no incluye todas las métricas del producto.

1.6 Conclusiones Parciales

Después de analizar exhaustivamente todos los conceptos relacionados con el tema a tratar en esta investigación, se puede asegurar el papel que juegan las métricas en los proyectos productivos logrando entender qué ocurre durante su desarrollo y mantenimiento, controlando qué es lo que ocurre en los proyectos productivos, para así mejorar los procesos y los productos.

A la hora de desarrollar una solución de *software* a medida, es imprescindible analizar convenientemente la forma de trabajar, los procesos y procedimientos. Mediante un buen proceso de medición se podrá llegar a obtener una buena herramienta de *software* que optimizará el rendimiento de los proyectos productivos, a partir de la implementación de las métricas del producto.



CAPITULO 2

CAPÍTULO 2: Tendencias y tecnologías actuales.

2.1 Introducción

El creciente desarrollo de las tecnologías hacen mayores las probabilidades de elegir herramientas que satisfagan las necesidades de los clientes, evaluando su utilidad y eficacia práctica. Todo este constante movimiento es indispensable para el desarrollo del *software*.

En el capítulo presente se espera hacer un estudio de las tendencias actuales de las tecnologías y seleccionar las herramientas más acertadas para cumplir los requerimientos de los clientes, por ejemplo: metodologías de desarrollo de *software* (MDS), lenguajes de programación, lenguaje de modelado y gestores de bases de datos. Teniendo en cuenta además, las que más se utilicen en los proyectos productivos de la Universidad de las Ciencias Informáticas.

2.2 Metodologías de Desarrollo de Software

El Desarrrollo de *Software* es una tarea difícil, debido a la continua evolución en las aplicaciones informáticas por lo que fue necesario el imponer cierta disciplina sobre todo el proceso para lograr mayor eficacia en el mismo, esperando obtener productos de *software* con la mayor calidad requerida.

Como respuesta a este problema es que empieza el surgimiento de las metodologías, a decir de Pressman: “las metodologías especifican **cómo** se debe hacer el software, los roles que desempeña cada **quién** durante el proceso, **cuándo** hacer cada actividad y **qué** se debe hacer. Están reconocidas además, como el conjunto de técnicas, herramientas, procedimientos y soporte documental que ayuda a un ingeniero de software a construir el software”. **(Pressman R. S., 2002)**

La aplicación de metodologías tradicionales, sin embargo, no brindan una solución óptima para los proyectos donde los requisitos no son reconocidos con exactitud o el entorno es volátil, pues no están concebidas para trabajar con incertidumbre.

Es por ello que surgieron otras metodologías que tratan de adaptarse a la realidad del desarrrollo de *software*, las cuales son conocidas como metodologías ágiles.

2.3 Metodologías robustas o pesadas.

Las metodologías tradicionales especialmente intervienen en el control del proceso. Estas establecen de forma rigurosa las actividades implicadas, los artefactos que se deben producir y las herramientas y notaciones que se usarán durante el proceso de desarrollo.

En este grupo entre las que más se emplean se encuentran:

Rational Unified Process (RUP)

Microsoft Solution Framework (MSF)

2.3.1 Proceso Unificado de Desarrollo

RUP[6] es una metodología de desarrollo orientada a objetos, creada con el objetivo de producir *software* con calidad, basada en el UML[7], pertenece a la familia de metodologías "pesadas", a causa de la gran cantidad de procesos y documentación que requiere; su ciclo de vida está caracterizado por:

- Dirigido por casos de usos.
- Centrado en la arquitectura.
- Iterativo e incremental.

[6] Proceso Unificado de Desarrollo
[7] Lenguaje Unificado de Modelado

Sus principales elementos son:

- Trabajadores (Quién)
- Actividades (Cómo)
- Artefactos (Qué)
- Flujo de actividades (Cuándo)

En esta metodología se han agrupado las actividades en 9 flujos de trabajo, siendo los 6 primeros flujos de ingeniería y los 3 últimos de apoyo. Además presenta 4 fases:

- Inicio
- Elaboración
- Costrucción
- Transición

2.3.2 Microsoft Solution Framework

Microsoft Solution Framework (MSF) van a ser prácticas que, de acuerdo al contexto de proyecto, dígase tamaño del equipo, frecuencia de entregas, etc.; serán más o menos recomendables de aplicar. Un grupo de guías para lograr que una solución en sistemas de información pueda ser finalizada exitosamente, rápidamente y reduciendo la cantidad de personas y riesgos.
Las características de MSF son:

- Adaptable
- Flexible
- Escalable
- Agnóstico a tecnologías

Divide el proceso de desarrollo en cinco fases:

- **"Visión:** en esta primera fase se pretende dar una visión general del proyecto, identificando las tareas y, los entregables que permiten al equipo a cumplir con los requerimientos y objetivos.

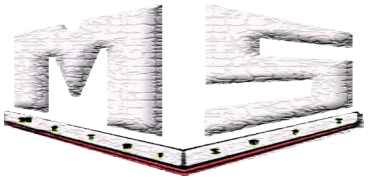
- **Planificación**: en esta fase se realiza la preparación de la especificación funcional, diseño de la solución, planes de trabajo, costes estimados y calendarios para los entregables. Implica la recogida y el análisis de los requerimientos de negocio, de usuario, operacionales y de sistema.

- **Desarrollo**: la meta de la fase de desarrollo es la construcción de los elementos y entregables de la solución, incluidos los códigos de los componentes, infraestructura (*software*, hardware, red) y la documentación para el uso de las operaciones.

- **Estabilización**: reproducción de condiciones reales y el equipo se concentra en detectar y priorizar errores, preparando la solución para su despliegue.

- **Distribución**: es la última fase de modelo. En la cual se instalan los componentes, se estabiliza el proyecto y se obtiene la aprobación por parte del cliente." **(López Requena, 2006)**

Figura 3: Fases MSF (López Requena, 2006)



CAPITULO 2

2.4 Metodologías ágiles.

Toda metodología debe ser adaptada al contexto del proyecto: recursos técnicos y humanos, tipo de sistema, tiempo de desarrollo, etc. Para hacerle frente con éxito a cualquier proyecto, no existe una metodología universal.

Una metodología ágil se destaca por su sencillez, tanto en su aplicación como en su aprendizaje, reduciendo de esta forma los costos de implantación en un equipo de desarrollo.

Se pueden resaltar como metodologías de este tipo:

Extreme Programming (XP)

RUP Ágil

RUP *Ultra Light*

2.4.1 Programación Extrema

Ideada por Kent Beck, XP[8] es un proceso de desarrollo de *software* diferente al convencional. Se caracteriza por ser flexible, poseer bajo riesgo y ser eficiente.

Se centra en en potenciar las relaciones entre cliente y equipo de desarrollo.Trata de dar al cliente el *software* que necesita y cuando lo necesita. Así como potenciar al máximo el trabajo en equipo.

"XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico." **(Canós, Letelier, & Penadés, 2005)**

Está definido principalmente por cuatro valores: comunicación, simplicidad, *Feedback* y coraje. Por lo que su misión principal es la revisión constante del código y la realización de forma frecuente de pruebas.

[8] **Programación Extrema**

Define además 13 prácticas:

1. “Equipo Completo
2. El Juego del *Planning*
3. Pequeños *releases*
4. Pruebas del Cliente
5. Propiedad Colectiva del Código
6. Estándares de codificación
7. Ritmo aceptable
8. Metáfora
9. Integración Continua
10. Desarrollo conducido por pruebas
11. *Refactoring*
12. Diseño simple
13. Programación a pares” **(Suma, 2008)**

2.4.2 RUP Ágil

“RUP no fue creado para ser una metodología pesada, aunque la gran cantidad de actividades y entregables opcionales que define ha llevado a la impresión o mala interpretación que sí lo es. En realidad fue definido para ser adaptado y aplicado en cada empresa de la forma que se crea necesario.” **(Suma, 2008).** RUP Ágil se divide en dos variantes:

- *Enterprise Unified Process* (EUP): es una versión extendida de RUP en la que se definen nuevas fases (Preconcepción, Producción y Jubilación) y 8 nuevas disciplinas basadas en directrices comunes de sistemas empresariales y de soporte.

- *Agile Unified Process* (AUP): es una versión simplificada de RUP, en la que se describe de una manera fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas (*test driven development* - TDD), Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de Base de Datos para mejorar la productividad.

2.4.3 RUP Ultra Light

RUP es un proceso muy organizativo, pero que requiere mucha documentación y para emplearlo es necesario adaptarlo a las necesidades de la entidad que lo utilice.

Con el fin de agilizar este proceso, muchas personas han tomado los elementos más significativos del mismo y es cuando empieza a surgir el término de RUP Ultra *Light*, esta metodología no es más que una adaptación del Proceso Unificado de Desarrollo de *Software* (RUP), integrada por 10 pasos enmarcados en 4 etapas: análisis, diseño, programación y puesta en marcha.

Se deben seguir los siguientes 10 pasos para desarrollar la MDS RUP Ultra Light:

1. **Realizar un Diagrama de Casos de Uso**. Cada caso de uso representa una funcionalidad del *software*, después de haberlas identificado, se representan los casos de uso en un diagrama, teniendo en cuenta los actores involucrados y las relaciones existentes entre ellos.

2. **Priorizar los Casos de Uso a trabajar**. Después de haber identificado los Casos de Uso, se les debe dar un orden de prioridad, donde la prioridad es el riesgo que conllevan. Los riesgos va a ser todo aquello que pueda afectar al sistema como no escoger bien los requerimientos, necesidad de

cambiar la arquitectura, etc. Se debe determinar los requerimientos más importantes e ir desarrollándolos por iteraciones.

3. **Generar los Documentos de Caso de Uso**. Para hacer entendible un diagrama de Casos de Uso, es necesario que el Analista del proyecto describa en un documento lo que hace el Caso de Uso, siguiendo más o menos la siguiente estructura:

    - "Descripción Breve. De lo que hace el Caso de Uso.
    - Precondiciones. Condiciones que deben ser cumplidas antes de ejecutar el Caso de Uso.
    - Flujo Básico. Descripción paso a paso de las acciones a realizar por el Usuario cuando trabaja de forma correcta en este Caso de Uso.
    - Flujo Alternativo. Detalle de los pasos que seguirá el usuario cuando no trabaje de forma correcta en este requerimiento, ejemplo: validaciones, etc.
    - PostCondiciones. Las acciones que se deben realizar después de que se ha terminado de ejecutar el Caso de Uso.
    - Interfaz Gráfica. Prototipo de cómo debe quedar la pantalla del caso de uso para ser vista por los usuarios." **(Guerrero, 2007)**

4. **Generar los Diagramas de Secuencia**. Los diagramas de Secuencia de UML permiten conocer la forma en la que los objetos se comunicarán en una pantalla para cumplir su objetivo. No es indispensable hacer este gráfico pero es muy recomendable pues ayuda a entender cómo se comunicarán los diferentes objetos entre sí, a través de los mensajes durante su ciclo de vida. Ayudando a que el Arquitecto de *Software* comprenda mejor lo que debe hacer.

5. **Diseñar el *FrameWork* del proyecto.** El Arquitecto de *Software* del proyecto va a diseñar las clases que se usarán en todo el Software. Este trabajo es bastante delicado ya que el mal diseño de las clases involucra que no se implementen las funcionalidades de cada clase de forma

correcta, lo cual conllevará a escribir un *"Spaghetti Code"*, que significa que el código estará enredado. Esta etapa de diseño debe ser revisada con cuidado y si es posible utilizar algún Patrón de Diseño de *Software*.

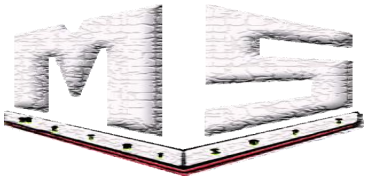
6. **Creación de la Base de Datos**. El diagrama de Clases de UML se establece como Base para el diseño de la Base de Datos del proyecto, de tal forma que utilicen la Capa de Datos, o sea, las clases de estereotipo Entidad.

7. **Construir la Aplicación *Desktop***. Mientras se van realizando los pasos 4, 5 y 6, el personal a cargo del diseño del sistema, puede ir creando los gráficos de las GUI's (*Graphic User Interface* o Interfaz Gráfica de Usuario), que s encuentran en los Documentos de los Casos de Uso.

8. **Programar las funcionalidades de los Casos de Uso**. Una vez terminadas las clases, se empezará a programar las funcionalidades de los Casos de Uso, para ello los programadores tomarán como referencia los documentos de Casos de Uso que los analistas desarrollaron y basándose en el Diagrama de Secuencia y en las clases diseñadas por el Arquitecto escribirán el código necesario para que el caso de uso funcione.

9. **Probar los Requerimientos del Software**. Independientemente que en el Documento de Casos de Uso se describa detalladamente cómo debe funcionar el requerimiento y que se hayan realizado los diagramas, siempre se escapan algunos detalles que se deben corregir en una etapa de pruebas exhaustivas, que no deben ser hechas por las mismas personas que programaron los casos de uso.

10. **Integrar los requerimientos concluidos**. Finalmente, es indispensable unir todo lo que los programadores hicieron de forma tal que el sistema funcione como un todo y ponerlo a disposición de los usuarios.



CAPITULO 2

2.5 Selección de la metodología de desarrollo de software que soporta la solución del problema.

Después de haber analizado las características de las metodologías de desarrollo de *software* aquí expuestas y teniendo en cuenta la agilidad que se requiere al implementar la herramienta propuesta en este trabajo de diploma, se hace necesario aplicar la MDS[9] RUP Ultra *Light*.

Es escogida dicha metodología debido a que esta se adapta a las condiciones de trabajo existentes. En este caso todo el equipo de desarrollo cuenta con dos personas que serán las encargadas de desempeñar todos los roles de Ingeniería de *Software* que implique la implementación de la herramienta a desarrollar.

Por lo que se hace necesario reducir el proceso de desarrollo esperando cumplir a través de la misma, con las expectativas de los clientes, respondiendo con sus necesidades y dar inicio a la modelación de lo que se quiere hacer y lo que el sistema debe tener. De esta forma RUP Ultra *Light* va a agilizar los procesos en versiones simplificadas.

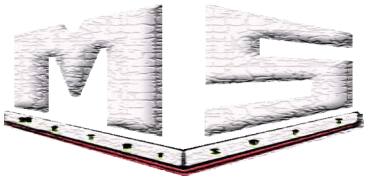
Dejar claro que con esta última se satisfacen las necesidades del cliente, se garantiza un proceso con una adecuada calidad (ya que la calidad del proceso repercute inevitablemente en la calidad del producto).

2.6 Selección de las herramientas de desarrollo.

Se propone desarrollar una Aplicación de Escritorio que facilite las operaciones de almacenamiento de los datos arrojados por las mediciones realizadas en los proyectos productivos de la Facultad 9 y el cálculo de las métricas del producto empleadas en dichos proyectos.

Aplicación de escritorio que está pensada en dos desarrolladores de cada proyecto productivo como máximo, el Administrador de la Calidad y el Responsable de la Medición. Por lo que se considera no sea necesario un sitio web al que tenga acceso todo el personal ajeno a la información allí expuesta, además de que no va a tener los permisos necesarios para acceder a ella. De ahí que la aplicación solo la van a instalar las personas capacitadas para interactuar con ella, haciendo más factible su uso.

[9] **Metodología de Desarrollo de Software**



Una herramienta sencilla y factible que responda a las necesidades de los equipos de Aseguramiento de la Calidad del *Software* (SQA) de cada proyecto productivo de la facultad y sirva como fuente de referencia para registros históricos que cuantifiquen el avance del desarrollo de cada proyecto en sí, siempre de acuerdo a las necesidades del cliente.

La aplicación que se va a desarrollar como parte de la propuesta solución tiene que cumplir con el propósito inicial de ser libre y multiplataforma de acuerdo a las factibilidades que proporcionan estos aspectos tanto al desarrollador como al usuario. Además de ser una aplicación amigable que posibilite un mejor manejo de los datos, no se necesita servidor y como aplicación pesada debe correr en el lado del cliente.

Para poder seleccionar las herramientas que se van a emplear durante el desarrollo de la aplicación, se realizará un estudio de las principales tecnologías utilizadas en la UCI como pueden ser: los lenguajes de programación y de modelado, así como de algunos de los sistemas gestores de bases de datos.

2.6.1 Lenguaje Unificado de Modelado (UML): soporte a la modelación de la propuesta de solución.

“Es un lenguaje que permite visualizar, especificar, construir y documentar los artefactos de los sistemas de software.” **(Larman, 1999)**
Precisamente esta definición aporta las principales funciones de UML:

- Visualizar: Expresa de forma gráfica un sistema de forma que sea fácil de comprender.
- Especificar: Especifica cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Permite la modelación de sistemas con tecnología orientada a objetos. Un modelo UML describe lo que supuestamente hará un sistema pero no dice como implementar dicho sistema.

En él se identifican:

- Elementos: estructurales, comportamiento, agrupación, anotación.
- Relaciones: dependencia, asociación, generalización/especialización, realización.
- Diagramas: clases, objetos, casos de uso, secuencia, colaboración, estados, actividades, componentes, despliegue.

2.6.2 Visual Paradigm: Herramienta CASE de la propuesta de solución

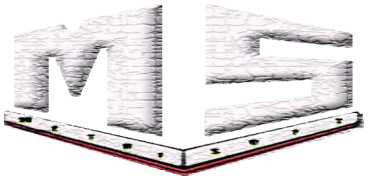
Para modelar los diagramas propios del desarrollo de la herramienta a utilizar se utilizará la herramienta *Visual Paradigm Enterprise* 6.4. Es una herramienta CASE[10] que utiliza "UML": como lenguaje de modelaje.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientado a objetos, construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

2.7 Lenguajes de Programación

Un lenguaje de programación permite crear programas para controlar el comportamiento físico y lógico de una máquina. Su estructura está definida por un conjunto de símbolos y reglas sintácticas y semánticas, que le dan significado a sus elementos y sus expresiones. Se pueden encontrar lenguajes de programación que permiten implementar tanto aplicaciones de escritorio como aplicaciones web.

[10] **Ingeniería de Software Asistida por Ordenador**



2.7.1 C++

C++ es un lenguaje de programación desarrollado a mediados de los años 1980 por Bjarne Stroustrup. Es una extensión del lenguaje C permitiendo la manipulación de objetos, o sea, su programación es orientada a objetos. Es un lenguaje híbrido. Tiene asociado a él particularidades como: sobrecarga de operadores, herencia múltiple, creación de clases abstractas, etc.

Contiene los paradigmas de programación estructurada, programación orientada a objetos y programación genérica, de ahí que se reconozca como un lenguaje multiparadigma. C++ permite trabajar tanto a alto como a bajo nivel.

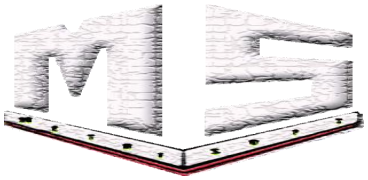
2.7.2 CSharp (C#)

Csharp es el lenguaje nativo de .NET (interfaz de programación de aplicaciones). Diseñado por *Microsoft* para su plataforma .NET. Es un lenguaje de programación de alto nivel orientado a objetos. Se deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET el cual es similar al de *Java* aunque incluye mejoras derivadas de otros.

Es actualmente uno de los lenguajes de programación más populares en la informática y las comunicaciones. El objetivo de *Microsoft*, es permitir a los programadores abordar el desarrollo de aplicaciones complejas con facilidad y rapidez. Se toman todas las cosas buenas de *Visual Basic* y se añaden a C#, aunque recortando algunas de las tradiciones más ocultas y difíciles de conocer de C y C++. Con C# no sólo se pueden escribir programas para la *Web*, sino que también permite desarrollar aplicaciones de propósito general.

2.7.3 Java

Java fue creado para abrir una nueva vía en la gestión de *software* complejo; es un lenguaje de programación orientado a objetos desarrollado a principios de los años 90 por *Sun Microsystems*.



La sintaxis de *Java* procede en gran medida de C++. Pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, *Java* fue construido desde el principio para ser completamente orientado a objetos. Todo en *Java* es un objeto (salvo algunas excepciones), y todo en *Java* reside en alguna clase.

Entre noviembre de 2006 y mayo 2007 *Sun Microsystems* liberó la mayor parte de sus tecnologías *Java* bajo la licencia GNU GPL, por lo que en la actualidad todo el paquete *Java* es libre aunque la biblioteca de clases para ejecutar los programas no lo es aún.

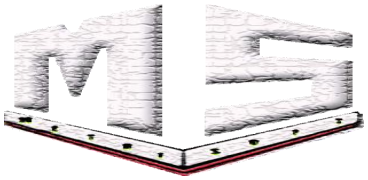
Java es un lenguaje de alto nivel, orientado a objetos, simple, dinámico, interpretado, seguro, robusto y portable que proporciona librerías y herramientas con el fin de que los programas puedan correr en varias máquinas de forma interactiva.

2.8 Selección del lenguaje de programación a utilizar en la implementación de la herramienta a desarrollar

Con la realización de los epígrafes anteriores, se hizo un esbozo de los principales lenguajes de programación que se emplean internacionalmente, en la UCI y por tanto, en la Facultad 9. Por lo que se procede a definir el lenguaje más apropiado para desarrollar la aplicación de escritorio que se desea implementar.

Los lenguajes mencionados con anterioridad, presentan entre sí características que los hacen comunes y a la vez presentan mejoras en la medida en la que los desarrolladores van trabajando en pos de hacer lenguajes más eficientes.

Para la implementación de la herramienta que se desea desarrollar, se escoge el lenguaje de programación Java, debido a que las aplicaciones de Java se distinguen por su extrema seguridad ya que no acceden a zonas delicadas de memoria o de sistema y posee un método súper seguro de autentificación por clave pública para evitar que los piratas informáticos desarrollen efectivas vías de intrusión.



La plataforma *Netbeans* es un proyecto de código abierto que permite desarrollar las aplicaciones mediante módulos. Estos módulos están formados por un archivo Java que les permite interactuar con las APIs de Netbeans y un archivo especial (*manifest file*) que es el que los identifica como módulos.

Las aplicaciones creadas a partir de módulos pueden ser extendidas añadiendo nuevos módulos. El IDE *Netbeans GUI* adjunta las tecnologías Java Desktop, haciendo posible construir de forma fácil una aplicación de escritorio.

Este es el lenguaje más apropiado para aplicar en la implementación de la solución propuesta, debido a que con él se pueden crear productos de *software* multiplataforma y libres, eliminando las restricciones del *software* propietario.

2.9 Sistemas de Gestión de Bases de Datos (SGBD)

Un SGBD es un tipo de *software* que sirve de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Es un conjunto de programas que permiten la creación de bases de datos asegurando su integridad, confidencialidad y seguridad. Estos sistemas permiten la manipulación de grandes volúmenes de información así como el control de la redundancia.

2.9.1 MySQL

MySQL surge como idea de la empresa *opensource* MySQL AB con el objetivo de que cumpla el estándar SQL (Lenguaje de Consulta Estructurado). Es un sistema de gestión de base de datos relacional, esto significa que archiva los datos en tablas separadas en vez de colocar todos los datos en un gran archivo.

Es multihilo por lo que puede realizar varias tareas a la vez y, es también multiusuario, lo que le permite satisfacer de forma simultánea, las necesidades de varios usuarios que comparten los mismos recursos.

Es un *software* de código abierto por lo que cualquier persona puede usarlo y modificarlo. Funciona además sobre múltiples plataformas.

2.9.2 Postgres SQL

Es un sistema de gestión de base de datos orientada a objetos, libre y multiplataforma. Incluye herencia entre tablas e incorpora una estructura de datos *array*. Además es altamente extensible.

PostgresSQL es un gestor de base de datos objeto-relacional, que utiliza Control de Concurrencia Multi-Versión con el fin de evitar bloqueos innecesarios, esta estrategia permite obtener una mejor respuesta en ambientes de grandes volúmenes.

La flexibilidad del API[11] de PostgreSQL permite facilitar soporte para el desarrollo con este SGBD en varios lenguajes de programación: Object Pascal, Python, Perl, PHP, Java/JDBC, Ruby, C/C++.

Incluye también una extensión que le añade soporte de objetos geográficos, PostGIS, que permite realizar análisis mediante consultas SQL espaciales o mediante conexión a aplicaciones GIS.

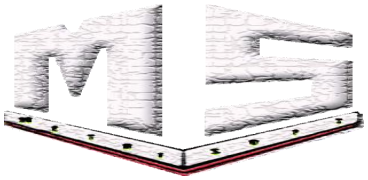
2.10 Selección del SGBD a utilizar

El SGBD escogido para dar solución a la aplicación Desktop a desarrollar es PostgresSQL.

Justificando que *PostgreSQL* es un SGBD libre y multiplataforma que centra su estrategia en un sistema llamado MVCC (*Multi Version Concurrent Access*) o Acceso Concurrente Multiversión, este da la posibilidad de que mientras un proceso esté escribiendo una tabla otros pueden acceder a la misma tabla, obteniendo de esta forma siempre una versión consistente de lo último a lo que se le hizo un cambio.

El mismo contiene características que permiten conectarlo fácilmente al IDE *NetBeans* 6.8 (herramienta donde se implementará la propuesta de solución), posibilita las prácticas de manejo y control de datos, es fácil para el aprendizaje y uso de los desarrolladores y está considerado como uno de los SGBD más actualizado.

[11] **Interfaz de programación de aplicaciones**



CAPITULO 2

2.11 Arquitectura de Software

"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución." Según el documento de IEEE Std 1471-2000 y adoptada también por Microsoft.

Se puede decir entonces que la Arquitectura de *Software* indica la estructura, funcionamiento e interacción entre las partes del *software*, es decir, es el diseño de más alto nivel de la estructura de un sistema.

2.11.1 Estilos arquitectónicos

Los estilos, expresan la arquitectura en el sentido más formal y teórico. Se asemejan "a descripciones informales de arquitecturas basadas en una colección de componentes computacionales, junto a una colección de conectores que describen las interacciones entre los componentes." **(Garlan & Allen, 1996).**

Un estilo define una forma de organización arquitectónica y conjugan elementos (componentes), conectores, configuraciones y restricciones. Entre los estilos arquitectónicos más utilizados en la actualidad se encuentra el estilo de llamada y retorno, el cual se empleará para lograr la implementación de la herramineta que se desea desarrrollar.

2.11.2 Estilos de Llamada y Retorno

"Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de esta familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas." **(Reynoso, 2004)**

Esta familia de estilos se divide en:

2.11.2.1 Modelo Vista Controlador (MVC):

"Al estilo MVC en ocasiones suele definírsele como un patrón de diseño y consiste en separar el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes" **(Burbeck, 1997):**

- Modelo: Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada. El modelo debe de preservar la inegridad de los datos.

- Vista: Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

- Controlador: Recibe las entradas, usualmente como eventos, e interpreta las operaciones del usuario; codificando los movimientos, pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista. Es el que debe de controlar los eventos.

2.11.2.2 Arquitectura en Capas:

Se define el estilo en capas como "una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción."**(Shaw & Mary, 1994)**

Este estilo incluye en sus restricciones topológicas, una limitación que exige que cada capa debe operar solamente con las capas adyacentes, así como los elementos de una capa deben entenderse sólo con elementos de la misma. Se plantea que si esta exigencia se relaja, el estilo deja de ser puro y pierde algo de su capacidad heurística. Respecto a este tema hay numerosos criterios en la literatura especializada,

tanto a favor como en contra, cada uno de ellos con argumentos válidos, aunque en su mayoría se plantea que la pureza de la arquitectura en capas puede sacrificarse siempre que sea en función de mejorarla.

2.12 Selección de la arquitectura que se va a utilizar

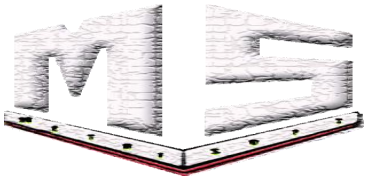
Para el diseño de aplicaciones con sofisticadas interfaces se utiliza el patrón arquitectónico Modelo-Vista-Controlador. Los proyectos productivos de la Facultad 9, en su mayoría utilizan el modelo arquitectónico MVC, por lo que el modelo arquitectónico a utilizar en la propuesta solución sería el mismo, pues la principal ventaja de las separaciones reside en la facilidad para realizar cambios en la aplicación puesto que cuando se realiza un cambio de base de datos, programación o interfaz de usuario, solo se tocará alguno de los componentes; se puede además modificar uno de los componentes sin conocer cómo funcionan los otros.

La variante que se va a emplear en la propuesta desarrolla una comunicación entre el Modelo y la Vista, donde esta última al mostrar los datos los busca directamente en el Modelo, dada una indicación del Controlador, disminuyendo el conjunto de responsabilidades de este último. Se trata de realizar un diseño que desacople la Vista del Modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.

Haciendo posible de esta forma que sus vistas siempre muestren información actualizada, dejando al programador libre de preocupaciones de solicitar que las vistas se actualicen, pues este proceso es realizado automáticamente por el modelo de la aplicación.

2.13 Framework

Un *framework* es la representación de una arquitectura de *software* que se adapta a las particularidades de un determinado dominio de aplicación, proporcionando una estructura y una metodología de trabajo que van a utilizar las aplicaciones de dicho dominio.



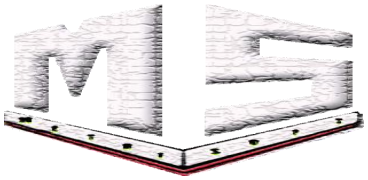
“En términos generales, se puede decir que un *framework* es un conjunto de clases que cooperan y forman un diseño reutilizable para un tipo específico de *software*. Un *framework* ofrece una guía arquitectónica partiendo el diseño en clases abstractas y definiendo sus responsabilidades y sus colaboraciones. Un desarrollador personaliza el *framework* para una aplicación particular mediante herencia y composición de instancias de las clases del *framework*.” **(Ganma)**

2.12.1 Hibernate

El *hibernate* es la herramienta de Mapeo objeto-relacional que se va a emplear en la propuesta solución que tiende a ser una solución completa al problema de los datos persistentes en Java, mediando la interacción de la aplicación con una base de datos relacional, brindando al desarrollador libertad para concentrarse en los problemas relacionados con la lógica del negocio.

Con *hibernate* se pretende solucionar las diferencias existentes entre el usado en la memoria de la computadora y el usado en las bases de datos, permitiendo al programador definir cómo es su modelo de datos, las relaciones que existen y qué forma tienen. Haciendo posible manipular los datos de la base operando sobre objetos, con todas las características de la Programación Orientada a Objetos (POO).

Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

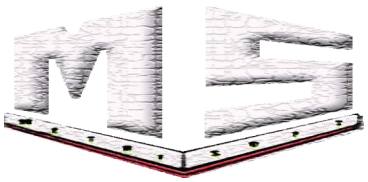


CAPITULO 2

2.13 Conclusiones Parciales

Todo el estudio y selección de herramientas representados anteriormente se efectuó con el objetivo de dar cumplimiento a la solución propuesta y a los lineamientos establecidos como soporte en la UCI se concluye que:

- Se decide utilizar RUP Ultra *Light* como MDS, por su factibilidad y características expuestas de antemano.
- Hacer uso del lenguaje de modelado UMLv2.0 y, herramienta CASE *Visual Paradigm v6.4 Enterprise Edition* para el modelado de los artefactos generados durante la modelación y, construcción de la herramienta que se desea implementar.
- El IDE *NetBeans* 6.8 será la plataforma de desarrollo para implementar la propuesta de solución por su flexibilidad y potencia. Además de utilizar en su entorno el lenguaje de programación Java -seleccionado para programar la aplicación- y, es una herramienta libre y multiplataforma; cumpliendo con las políticas de producción establecida en la Facultad 9.
- El SGBD seleccionado es *PostgreSQL v*8.4 pues tiene un alto rendimiento y procesamiento de datos, también presenta grandes facilidades de integración con el NetBeans y Java como lenguaje de programación.
- El estilo arquitectónico que se decide utilizar es el MVC.
- Los *Frameworks* ayudan en el desarrollo de *software*, proporcionando una estructura definida la cual ayuda a crear aplicaciones con mayor rapidez, por lo que para dar solución al problema planteado se seleccionó el *Hibernate v2.1* como *framework* a utilizar en la propuesta.



CAPÍTULO 3: Presentación de la solución propuesta.

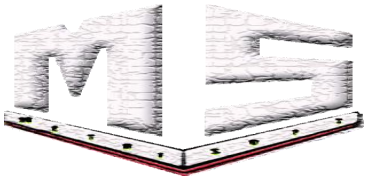
3.1 Introducción

Este capítulo va a presentar las características que debe tener y debe cumplir el sistema que se quiere desarrollar. Utilizando la Metodología de Desarrollo de *Software* seleccionada en el capítulo anterior -RUP Ultra Light- ; comenzando el proceso con la modelación de la aplicación de escritorio a partir de la captura de los requerimientos funcionales y no funcionales, seleccionando los actores que van a interactuar con el sistema, el modelo de casos de uso que se va a utilizar, así como sus relaciones. Además de las descripciones textuales de los diferentes casos de uso definidos.

3.2 Descripción del Sistema Propuesto

Se ha concebido como propuesta de solución al problema planteado, la implementación de una aplicación que garantice automatizar la medición de productos de *software* en los proyectos productivos de la Facultad 9. El sistema le proporcionará a los especialistas una herramienta necesaria para lograr mayor rapidez en su trabajo.

En la etapa de Inicio de la investigación, se efectuaron encuestas y entrevistas a los respectivos jefes de Departamentos de Centro de la Facultad 9, donde se detectó que la mitad de los jefes de departamentos cuenta un conocimiento parcial del concepto de métricas, así como de los procesos de medición y control de la calidad a partir de las métricas definidas por la Dirección de Calidad en la UCI, por lo que no se rigen por este modelo de calidad.



CAPITULO 3

Todos los proyectos productivos de la Facultad 9 presentan un Plan de Aseguramiento de la Calidad que debería estar a cargo de las buenas prácticas de los procesos de medición y evaluación de la calidad, y debido a que no tienen una estructura organizacional adecuada en la asignación de una persona al rol de la medición dentro del proyecto, ya sea porque cada persona se desempeña en diferentes roles -complejos o no- esta situación hace que existan divergencias en los propósitos planteados por el área de proceso de Medición y Análisis de CMMI y la Metodología de Desarrollo RUP empleada en la mayoría de los proyectos productivos de la Facultad 9.

De acuerdo a los resultados arrojados por las encuestas y entrevistas realizadas se puede afirmar entonces, que en el entorno de los clientes debido a que no está definido en el ámbito del proyecto el proceso de medición del *software*, se decide no realizar modelamiento de negocio, sino que estaría determinado por la descripción de un Modelo de Dominio por la ausencia de los procesos de medición del *software*, los que son necesarios para la modelación.

Para dar solución a esta situación, se pasa directamente a la modelación de la herramienta a partir de la Captura de Requisitos, obviando la etapa de Modelamiento de Negocio, dando soporte a la selección de la MDS RUP Ultra *Light* especificada en el capítulo anterior.

3.2.1 Requerimientos del sistema a implementar.

La captura de los requisitos es una de las actividades fundamentales en la fase de inicio de la modelación de un *software*, son una descripción de las necesidades o deseos de un producto, teniendo como objetivos principales:

- “Establecer y mantener un acuerdo con los clientes y otros interesados acerca de lo que debe hacer el sistema.
- Proporcionar desarrolladores de sistema con un buen conocimiento de los requisitos del sistema.
- Definir los límites del sistema (delimitarlo).
- Proporcionar una base para planificar el contenido técnico de las iteraciones.
- Proporcionar una base para la estimación del coste y del tiempo en que desarrollar el sistema.

- Definir una interfaz de usuario para el sistema, centrándose en las necesidades y los objetivos de los usuarios.” **(RUP, 2003)**

Los requerimientos deben ser validados y verificados para evitar futuros errores en el desarrollo de la herramienta que se modela, siendo estos, la base de los casos de uso del sistema (CUS).

Partiendo del estudio de los procesos de medición del *software* y el apoyo en el área de procesos de Medición y Análisis de CMMI se identificaron los siguientes requerimientos funcionales y no funcionales.

3.2.1.1 Requerimientos Funcionales (RF)

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir.

R1 Autenticar Usuario

R1.1 Cambiar contraseña

R2 Gestionar usuario

R2.1 Insertar Usuario

R2.2 Eliminar Usuario

R2.3 Modificar Usuario

R3 Gestionar Proyecto

R3.1 Insertar datos de proyecto

R3.2 Modificar datos de proyecto

R3.3 Eliminar datos de proyecto

R4 Gestionar Departamento

R4.1 Insertar datos de Departamento

R4.2 Modificar datos de Departamento

R4.3 Eliminar datos de Departamento

R5 Gestionar datos de Métricas

R5.1 Insertar datos de Métricas

R5.1.1 Calcular datos de Métricas

R5.2 Eliminar datos de Métricas

R5.3 Modificar datos de Métricas

R6 Generar Reporte

R6.1 Obtener Reporte

R6.2 Exportar Reporte a PDF

R6.3 Imprimir Reporte

R6.4 Guardar Reporte

R7 Obtener historial de proyecto por métricas

3.2.1.2 Requerimientos No Funcionales (RNF)

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable, buscando lograr una buena aceptación del cliente para con el producto.

El sistema debe tener:

RNF 1 Apariencia o interfaz externa:

1.1 El diseño de la aplicación de escritorio debe ser sencillo para evitar dificultades en el momento de su uso. Debe estar compuesto por interfaces gráficas independientes de acuerdo a la funcionalidad seleccionada, ventanas de errores, formularios de datos y otros componentes visuales.

1.2 La aplicación debe ser atractiva al usuario y presentar colores contrastantes que brinden confianza a los usuarios en el momento en que interactúen con ella.

1.3 La dimensión del diseño visual de la herramienta estará definida bajo la resolución 800 x 600 o en resoluciones mayores.

RNF 2 Confiabilidad:

El sistema debe realizar:

2.1 Un tratamiento de excepciones y validaciones en las entradas de usuarios.

2.2 El sistema debe ser confiable en el almacenamiento de los datos que procesa.

RNF 3 Rendimiento:

3.1 Tener base de datos normalizada, para garantizar la integridad de la información y reducir los tiempos de respuesta.

3.2 Permitir numerosas conexiones simultáneas.

RNF 4 Soporte:

4.1 Se requiere de un servidor de base de datos con las siguientes características:

- Velocidad de procesamiento: 100 Kb/s (mínimo)
- Tiempo de respuesta: 10 segundos (máximo)

RNF 5 Seguridad:

5.1 El usuario podrá acceder a la funcionalidad a la cual tenga acceso de acuerdo al rol que desempeñe.

5.2 Verificación de opciones del usuario ante acciones no reversibles (Modificación o Eliminación de datos) a través de ventanas de confirmación de acciones. Aceptar/ Cancelar.

RNF 6 Usabilidad:

El sistema podrá ser usado por un usuario con:

6.1 Conocimientos básicos en tecnología *Desktop* y el manejo de la computadora.

6.2 Conocimientos parciales o totales de los procesos de cálculo y recopilación de datos de las métricas de los productos.

RNF 7 Portabilidad:

7.1 El sistema debe ser independiente de la plataforma.

RNF 8 Software:

8.1 En el sistema se utilizará un servidor que soporte el Sistema Operativo: Windows, con el gestor de Base de Datos PostgreSQL 8.4 instalado.

8.2 Las computadoras de los clientes deben tener instalado la máquina virtual de java y Adobe Reader 9.10 instalado. (Para la lectura de documentos PDF)

RNF 9 Hardware:

9.1 Para instalar la aplicación la computadora de los clientes necesitan:

- Memoria RAM: capacidad de 512 Mb (mínimo) – 1Gb (recomendado).
- Disco Duro: de 40Gb (mínimo) – 80Gb (recomendado).
- Microprocesador: Intel, Dual-Core, Core-Dual

9.2 El servidor de Datos debe contener:

- Servidor de Base de Datos: 40Gb (mínimo)

3.3 Descripción de los actores

Después de haber detallado los requerimientos funcionales y no funcionales del sistema, se lleva a cabo entonces, la selección de los actores del sistema. Un actor del sistema representa a un tercero fuera del sistema que colabora con este.

Durante todo el proceso de evaluación y control de la calidad de los proyectos productivos que se realiza en la Facultad 9 interviene el Grupo de Calidad, donde los miembros del SQA responden a desarrollar ciertas estrategias de Pruebas, Auditorías, Revisiones, etc., logrando una mejora significativa en los mismos en cuanto a calidad se refiere.

La siguiente tabla muestra los actores del sistema que interactuarán con la aplicación que se desea implementar, actores que desempeñan ciertos roles dentro del SQA de cada proyecto productivo. Teniendo en cuenta que el sistema que aquí se modela, está pensado en los responsables del Plan de Aseguramiento de la Calidad y el rol responsable de la medición.

Actor	Justificación
Administrador de la Calidad	Responsable del equipo SQA que tiene los privilegios para gestionar los datos de los usuarios, así como el control del acceso para que puedan interactuar con las funcionalidades del sistema. Nota: El líder de proyecto puede comportarse como un responsable SQA en un momento determinado.
Responsable de la Medición	Miembro del equipo SQA de un proyecto productivo determinado encargado de realizar las actividades de cálculo y gestión de métricas, proyectos y obtención de historiales.

Tabla 2: Descripción de los actores del Sistema

3.4 Casos de Uso del Sistema

La forma en que interactúa cada actor del sistema con el sistema se representa con un Caso de Uso. Los Casos de Uso son los que conducen toda la arquitectura del sistema.

A continuación se representa el Modelo de Casos de Uso del Sistema (MCUS), modelo que contiene actores, casos de uso y sus relaciones.

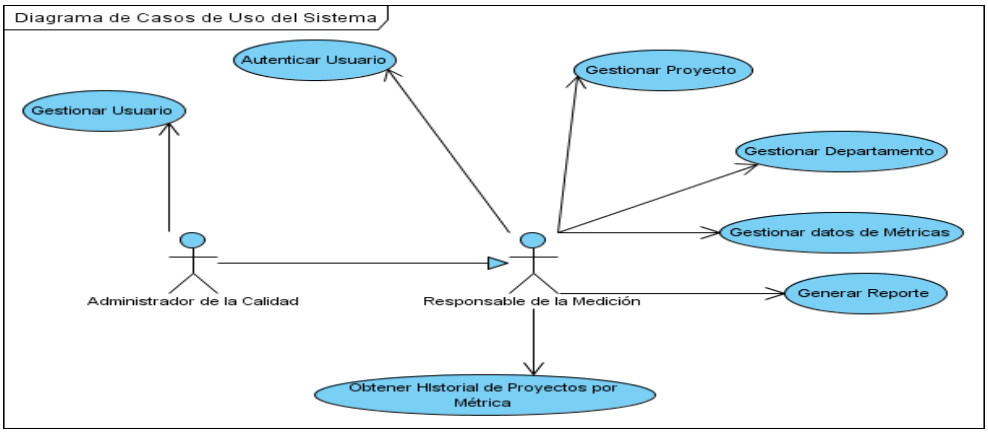
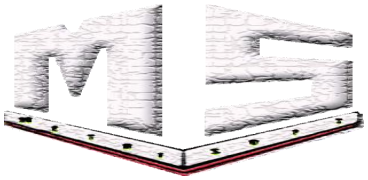


Figura 4: Modelo de Casos de Uso del Sistema.



CAPITULO 3

3.5 Descripción de los Casos de Uso del Sistema

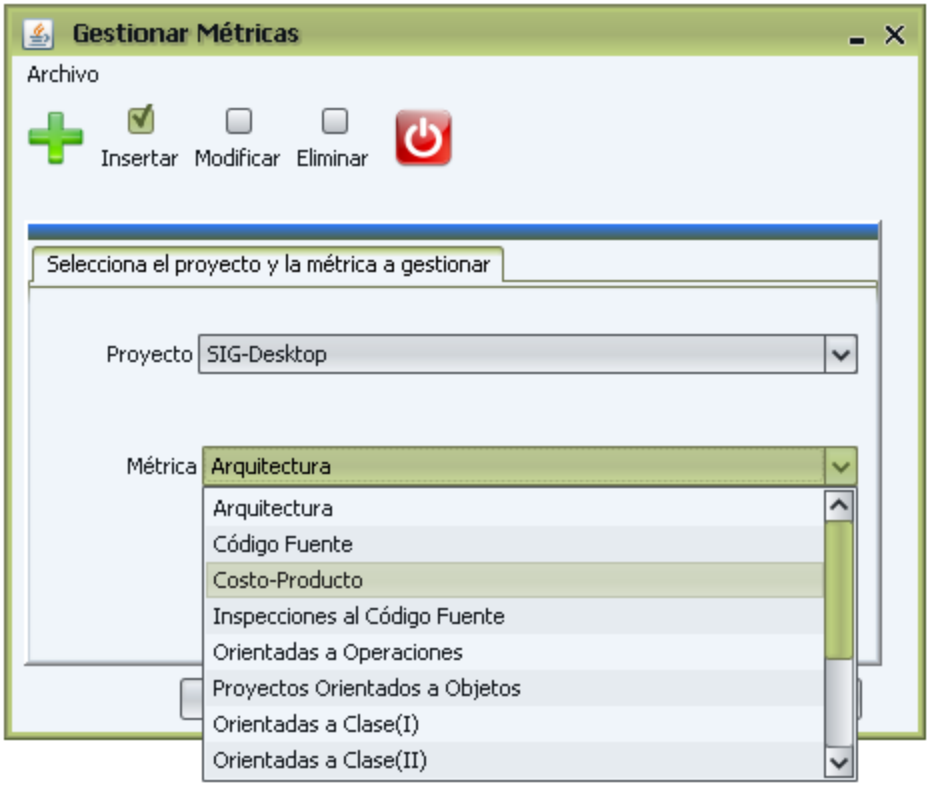
3.5.1 Descripción del CU_Gestionar datos de Métricas

Caso de Uso:	Gestionar datos de Métricas
Actores:	Responsable de la Medición
Resumen:	El CU inicia cuando un Responsable de la Medición desea insertar, eliminar y modificar una métrica.
Precondiciones:	Que el Responsable de la Medición esté autenticado. Debe haberse insertado al menos un proyecto al sistema.
Referencias	RF4, RF4.1,R4.1.1, RF4.2 y RF4.3
CU Asociados	
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. El Responsable de la Medición selecciona la opción de “Gestionar datos de la Métrica (Nombre de la métrica)”. 3. El Responsable de la Medición selecciona la opción deseada.	*2.* El sistema muestra una vista con las opciones: • *Insertar datos de Métricas* • *Calcular datos de métricas* • *Modificar datos de Métricas* • *Eliminar datos de Métricas* 4. El sistema ejecuta la opción seleccionada: • Si el Responsable de la Medición desea insertar los datos de una métrica, ir a Sección “Insertar datos de métricas.” • Si el Responsable de la Medición desea eliminar los datos de una métrica, ir a Sección “Eliminar datos de métricas.” • Si el Responsable de la Medición desea modificar los datos de una métrica, ir a Sección “Modificar datos de métricas.”

Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3. El responsable SQA presiona el botón “Cancelar”.	4. Volver al flujo normal de los eventos del CU_ Gestionar datos de Métricas.
Sección “Insertar datos de Métricas”	
Acción del Actor	**Respuesta del Sistema**

2. El Responsable de la Medición introduce los datos de la métrica seleccionada correctamente. 4. El Responsable de la Medición oprime el botón “Insertar”.	1. El sistema muestra un formulario con campos que contienen las variables correspondientes a la métrica seleccionada para insertar los datos. 3. El sistema verifica que los datos estén correctos. • Si la métrica es indirecta el sistema ir a Sección “Calcular datos de Métrica”. 5. El sistema ejecuta la acción seleccionada y muestra el mensaje: “Los valores de la métrica fueron insertados satisfactoriamente.” y actualiza la base de datos.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. El responsable SQA presiona el botón “Cancelar”.	3. Volver a la Sección “Insertar datos de Métricas” CU_ Gestionar datos de Métricas.
Sección “Calcular datos de Métricas”	
Acción del Actor	**Respuesta del Sistema**

1. El Responsable de la Medición oprime el botón “Calcular” para calcular las variables de la métrica en cuestión. 3. Volver a inciso 4. de la Sección “Insertar datos de métricas”.	2. El sistema ejecuta la acción seleccionada y muestra los resultados del cálculo de la métrica.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
1. El Responsable de la Medición no oprime el botón “Calcular” o deja de introducir algún dato.	2. El sistema muestra el mensaje de error: “No pueden existir campos vacíos”.
Sección “Modificar datos de Métricas”	
Acción del Actor	**Respuesta del Sistema**

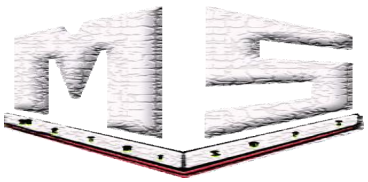
	1. El sistema muestra un listado con todas las métricas del producto escogidas.
2. El Responsable de la Medición selecciona la métrica a la cual quiere modificar los datos y presiona el botón “Aceptar”.	*3.* El sistema muestra un formulario con campos que contienen las variables correspondientes a la métrica seleccionada para modificar los datos.
4. El Responsable de la Medición modifica los datos deseados y presiona el botón “Modificar”.	5. El sistema muestra el mensaje “¿Está seguro que desea modificar los datos de la métrica seleccionados? “
6. El Responsable de la Medición presiona el botón “Aceptar”.	7. El sistema ejecuta la opción aceptada y actualiza los datos de la métrica correspondiente.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
6. El Responsable de la Medición presiona el botón “Cancelar”.	7. Volver a la Sección “Modificar datos de Métricas” del CU_ Gestionar Datos de Métricas.
Sección “Eliminar datos de métricas”	

Acción del Actor	Respuesta del Sistema
	1. El sistema muestra un listado con todas las métricas del producto escogidas.
2. El Responsable de la Medición selecciona la métrica a la cual va a eliminar los datos.	3. El sistema muestra el mensaje ¿Está seguro que desea eliminar los datos de la métrica seleccionada?
4. El Responsable de la Medición presiona el botón “Aceptar”.	5. El sistema ejecuta la opción aceptada y actualiza los datos.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
4. El Responsable de la Medición presiona el botón “Cancelar”.	5. Volver a la Sección “Eliminar datos de Métricas” del CU_ Gestionar Datos de Métricas.
PostCondiciones	Se logra insertar, modificar y eliminar los datos de las métricas del producto.

Tabla 3: Descripción del CU_Gestionar datos de Métricas

3.6 Conclusiones parciales

Al identificar los requerimientos funcionales y no funcionales, los actores del sistema, el modelo de casos de uso y sus descripciones textuales, se puede afirmar entonces que se va a realizar un diseño robusto de la aplicación que se desea implementar. Diseño que va a tener bien establecido las limitaciones que debe cumplir, de acuerdo a los requerimientos definidos; las funcionalidades que se van a automatizar son los diferentes casos de uso con sus especificaciones en las descripciones textuales de los mismos.



CAPÍTULO 4: Construcción de la solución propuesta.

4.1 Introducción

La arquitectura comprende a grandes rasgos, el conjunto de decisiones significativas sobre la organización del sistema y las interfaces que formarán parte de él, junto al tratamiento de la selección de los elementos estructurales. En este capítulo se crea entonces el espacio para dar una propuesta de solución acorde a las necesidades y expectativas del cliente de acuerdo a los requerimientos establecidos para el sistema en desarrollo.

Los aspectos relevantes del flujo de trabajo de Diseño de la herramienta seguido del proceso de confección de la base de datos que soportará dicho sistema, serán representados a continuación.

4.2 Conformación de la solución propuesta

La arquitectura es el balance entre arte e ingeniería y requiere de una aproximación mental específica para la resolución de problemas. La que se aplica para la realización de la herramienta propuesta se centra principalmente en los casos de usos arquitectónicamente significativos, siendo un objetivo principal del desarrollo del *software* seleccionar los requerimientos significativos para el sistema a implementar.

La arquitectura del sistema propuesto estará representada por las 4+1 vistas arquitectónicas, creadas por Philippe Krutchen en el año 1995. Están denominadas como vistas de: CU, Lógica, Implementación, Despliegue y Procesos.

4.3 Vista de Casos de Uso

Los Casos de Uso son una técnica para especificar el comportamiento y responder a las necesidades del sistema. La vista de Casos de Uso es la representación de los casos de uso arquitectónicamente más significativos; los que comprenden la base de la integridad y seguridad de la aplicación que se modela. Definen la arquitectura básica y tenerlos en cuenta implica saber con qué prioridad deben ser tratados y en qué orden deben ser implementados.

A la hora de elegirlos, es vital saber cuáles son los necesarios para la primera iteración en la etapa de desarrollo. Los resultados de este proceso se recogen en la siguiente vista:

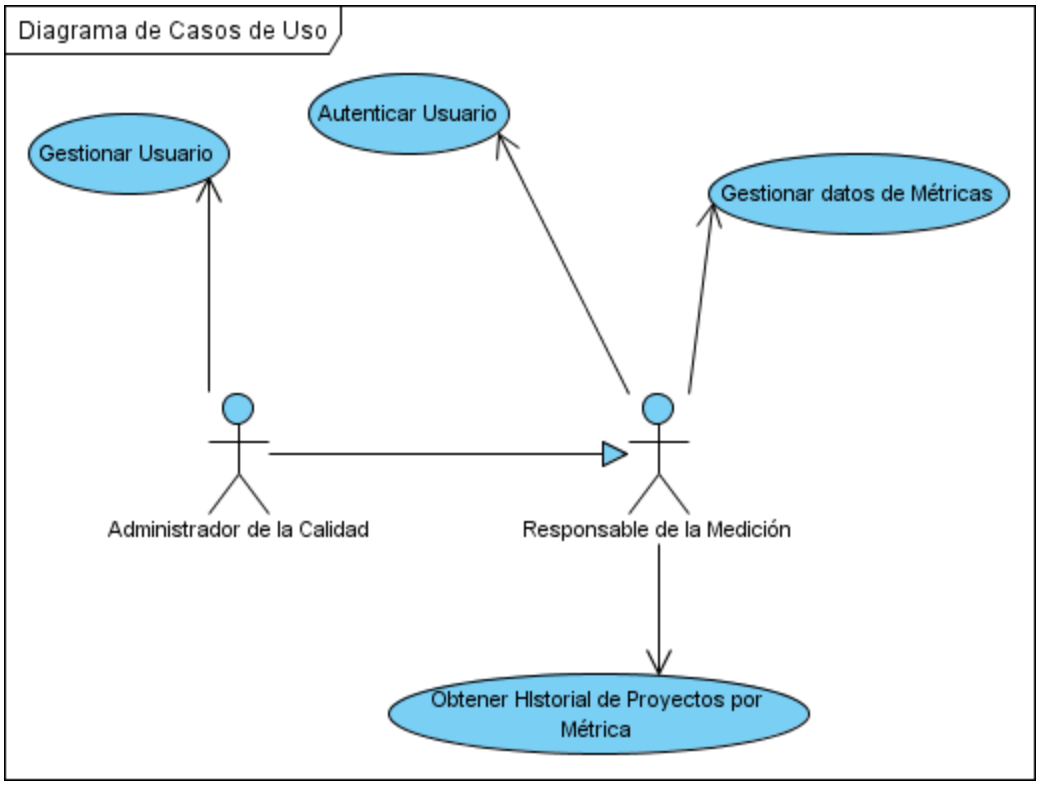


Figura 5: Vista de Casos de Uso

La vista anterior representa los casos de uso que se deben implementar tempranamente en la fase de construcción de la herramienta.

4.4 Vista Lógica

La vista lógica describe las clases más importantes identificadas durante el proceso de elaboración, como parte indispensable de la descripción de la arquitectura. Vista que se encarga de describir el diseño, el cual está muy cercano a la implementación.

La siguiente figura muestra la distribución en subsistemas que utiliza el Modelo Vista Controlador, patrón que se aplicará que es actualmente el más usado en la confección de aplicaciones debido a las ventajas que trae consigo como la forma en que organiza los elementos de la aplicación.

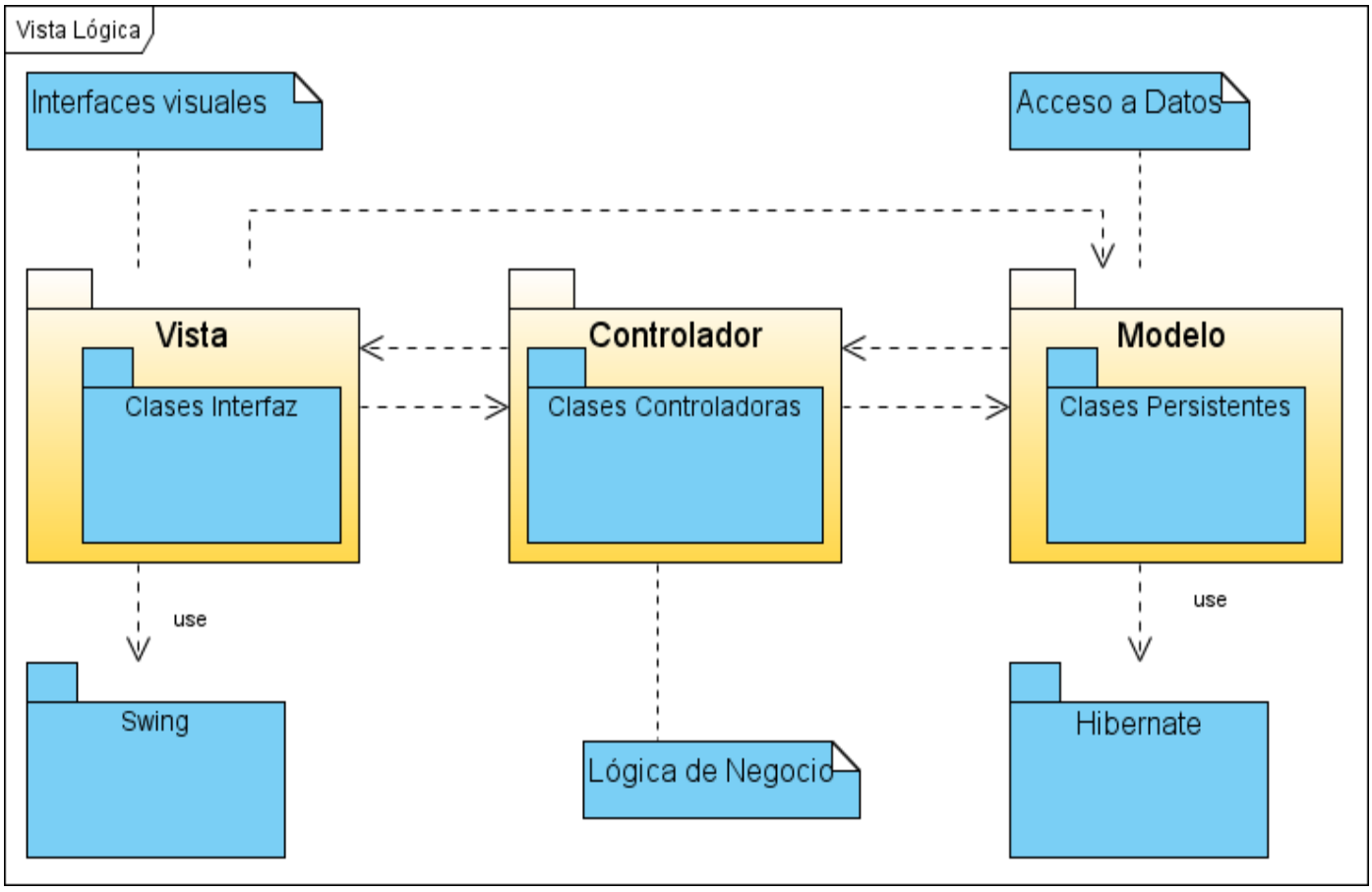
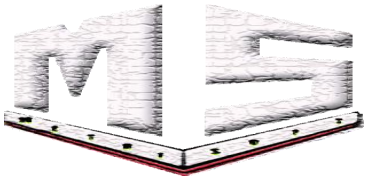


Figura 6: Vista Lógica



CAPITULO 4

El paquete Vista cuenta con todas las clases interfaz de usuario que se utilizarán en la vista presentada a los usuarios, para poder desarrollar estas vistas se debe contar con la biblioteca Swing que proporciona componentes visuales implementados, con los que se pueden realizar numerosas operaciones visuales y le ahorran tiempo al desarrollador pues solamente se deben reutilizar y configurar.

Su estructura está compuesta por:

- **Clases Interfaz**: Contiene todas las clases interfaz de usuario de cada uno de los CU arquitectónicamente significativos (Autenticar Usuario, Gestionar Usuario, Gestionar datos de Métricas, Obtener Historial de Proyectos por métrica).

En el paquete Controlador se reciben las peticiones del usuario, residen los programas que se ejecutan en el sistema y se establecen las reglas a cumplir. Es la Lógica de Negocio que contiene “todos los aspectos que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios”. **(Teruel, 2000).** Para enviar los resultados requiere de la comunicación con el paquete Vista para el recibo de las solicitudes de los usuarios, y para almacenar y recuperar datos de la Base de Datos se comunica con el paquete Modelo.

Su estructura está compuesta por:

- **Clase Controladora**: Contiene la clase controladora que contiene todo el comportamiento de las funcionalidades del sistema. En este caso las funcionalidades son los **CU críticos** (Autenticar Usuario, Gestionar Usuario, Gestionar datos de Métricas, Obtener Historial de Proyectos por métrica).

El paquete Modelo es el encargado del manejo de datos persistentes a través del uso de uno o más gestores de Base de Datos, los cuales funcionan a cargo del almacenamiento de los datos. Se comunica con el Controlador y recibe de él las solicitudes de almacenamiento o recuperación de información.Su estructura está compuesta por:

- **Clases Persistentes**: Contiene las clases persistentes a cargo manejo de datos de usuarios, proyectos, métricas, etc.

- ***Framework Hibernate***: Contiene los componentes del *framework Hibernate*.

4.5 Patrones de Diseño empleados

Los patrones de diseño “son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos” **(Gracia, 2005)**

Presentan características propias y abstractas que hacen de ellos una práctica difícil de entender, “pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Han revolucionado el diseño orientado a objetos y todo buen arquitecto de *software* debería conocerlos”. **(Gracia, 2005)**

Para lograr la estructura adecuada del sistema se hace uso de los Patrones GRASP de asignación de responsabilidades: Bajo Acoplamiento, Creador y Controlador.

4.5.1 Bajo Acoplamiento

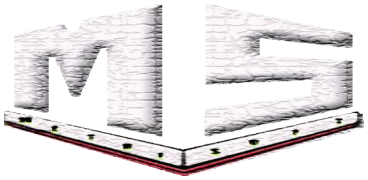
Es la idea de tener las clases lo menos ligadas entre sí que se pueda y cuando se logra un Bajo Acoplamiento en los diferentes módulos de un sistema se puede asegurar la buena distribución de las partes que intervienen en el mismo. Haciendo referencia a la acción de cada objeto de tener la mínima dependencia posible con el resto del sistema, posibilitando la realización de modificaciones en ciertas partes del programa sin que se afecten en gran medida los objetos relacionados.

4.5.2 Creador

El patrón Creador ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Brinda soporte a un bajo acoplamiento y mejores oportunidades de reutilización.

4.5.3 Controlador

El patrón Controlador sirve como intermediario a partir de una clase controladora entre las clases representadas en el paquete Vista y con cada clase persistente del Modelo. Sugiriendo la reutilización del código y un mayor control.



CAPITULO 4

4.6 Diagrama de Clases del Diseño

En la fase de diseño se modela el sistema de manera que soporte todos los requisitos, tanto funcionales como no funcionales, creándose así una entrada apropiada para las actividades de implementación.

El DCD[12] no es más que la vista estática del diseño del sistema. El mismo está compuesto por clases interfaces, controladoras y entidades.

4.7 Diseño de la Base de Datos

Una BD[13] es un conjunto de datos relacionados entre sí. Estas se transforman a un modelo de datos relacional basado en una estructura de datos simple y uniforme, y en sus relaciones, representando a la BD como una colección de relaciones.

El trabajar con *software* orientado a objetos y bases de datos relacionales puede hacer al implementador invertir mucho tiempo en los entornos actuales. Por ello en la propuesta solución se utiliza el *Framework Hibernate:* la herramienta que va a realizar el mapeo existente entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relación de las bases de datos; proporcionando servicios de persistencia y objetos persistentes a la aplicación que reducen el tiempo de desarrollo.

Con el *Hibernate* v2.1 el uso de la BD es completamente transparente pudiendo cambiar de la misma sin necesidad de cambiar una línea de código, simplemente cambiando los ficheros de configuración del *framework*.

A continuación se muestra el Diagrama de Clases Persistentes.

[12] Diagrama de Clases del Diseño
[13] Base de Datos

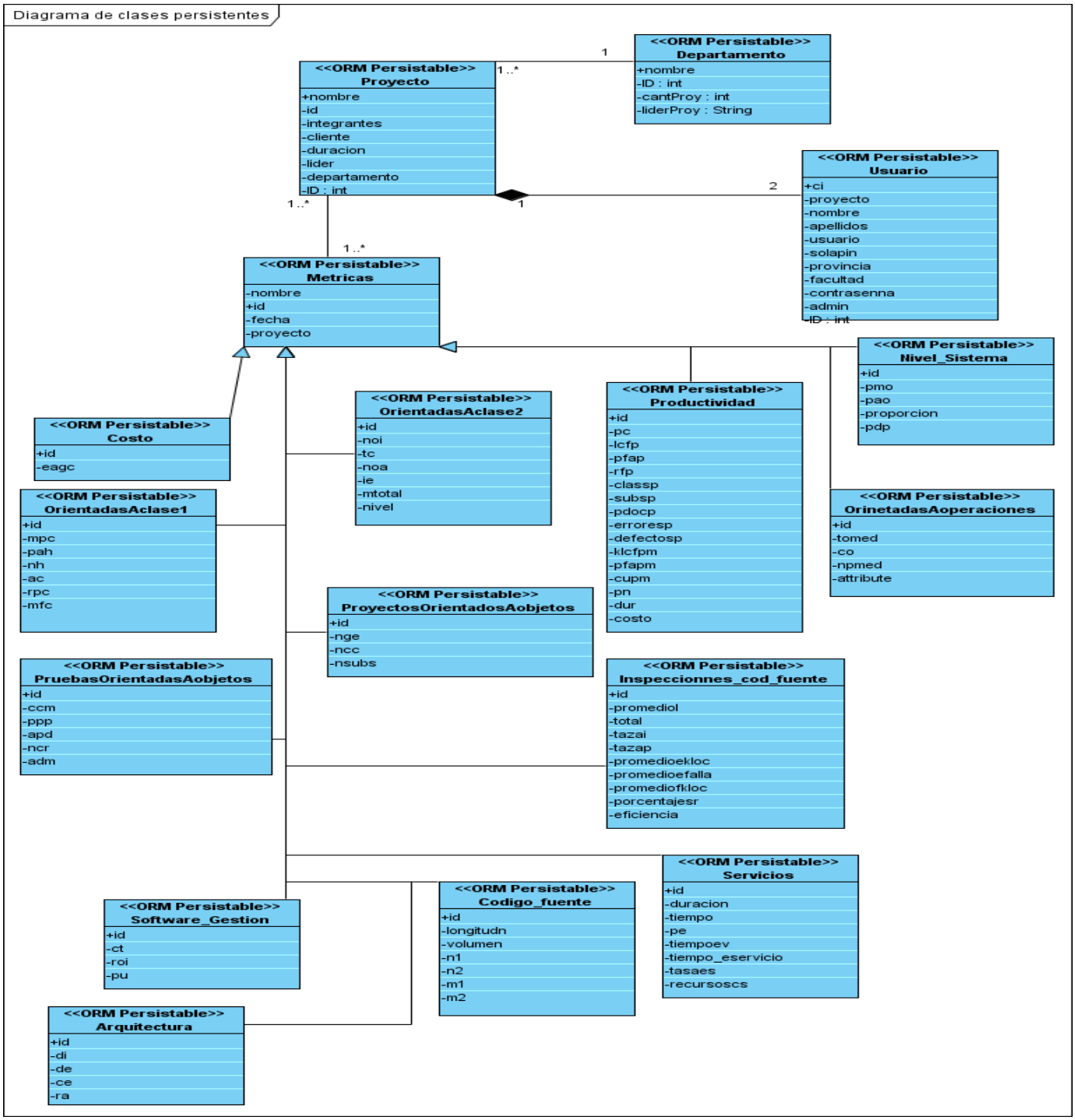


Figura 7: Diagrama de Clases Persistentes.

CAPITULO 4

4.8 Generalidades de la Implementación

En el presente epígrafe se abordarán aspectos generales de la implementación de la propuesta de solución. Mostrando las Vistas de Despliegue e Implementación por las cuales se regirá la construcción de la herramienta de *software* en desarrollo.

4.8.1 Vista de Despliegue

El Modelo de Despliegue o Vista de Despliegue "se utiliza para visualizar la distribución de los componentes de *software* en los nodos físicos" **(Piñeiro, 2009)** centrándose en los RNF[14] como la disponibilidad del sistema, la fiabilidad (tolerancia a fallos), ejecución y escalabilidad.

Captura "la configuración de los elementos de procesamiento, y las conexiones entre estos elementos en el sistema. Está compuesto por uno o más nodos, dispositivos y conectores. El modelo de despliegue también mapea procesos dentro de estos elementos de procesamiento, permitiendo la distribución del comportamiento a través de los nodos que son representados." **(Piñeiro, 2009)**

Es la forma de ubicar la herramienta de *software* que se construye en el hardware que la soporta

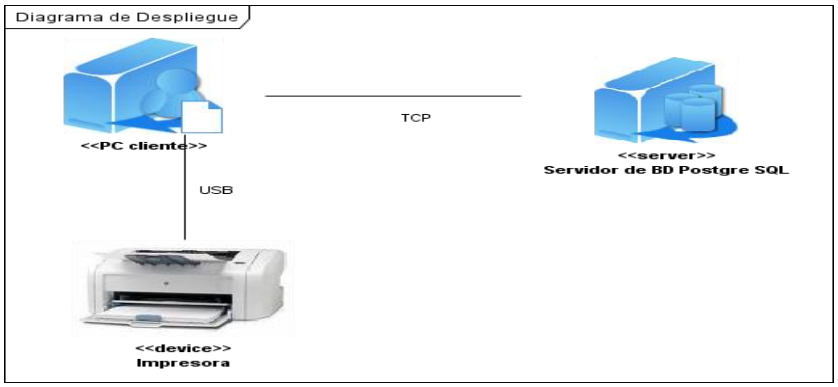


Figura 8: Vista de Despliegue.

[14] **Requisitos No Funcionales**

4.8.2 Vista de Implementación

La implementación se empezará con el resultado del diseño y se implementará el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares.

Un componente de *software* es esa parte física de un sistema que puede ser un módulo, una base de datos, un programa ejecutable, etc. Las clases son conceptos de abstracción que poseen atributos y métodos que se implementan o materializan en los componentes.

A continuación se presentan algunos de los diagramas de componentes por casos de uso, resultado de la vista de implementación de la aplicación a desarrollar:

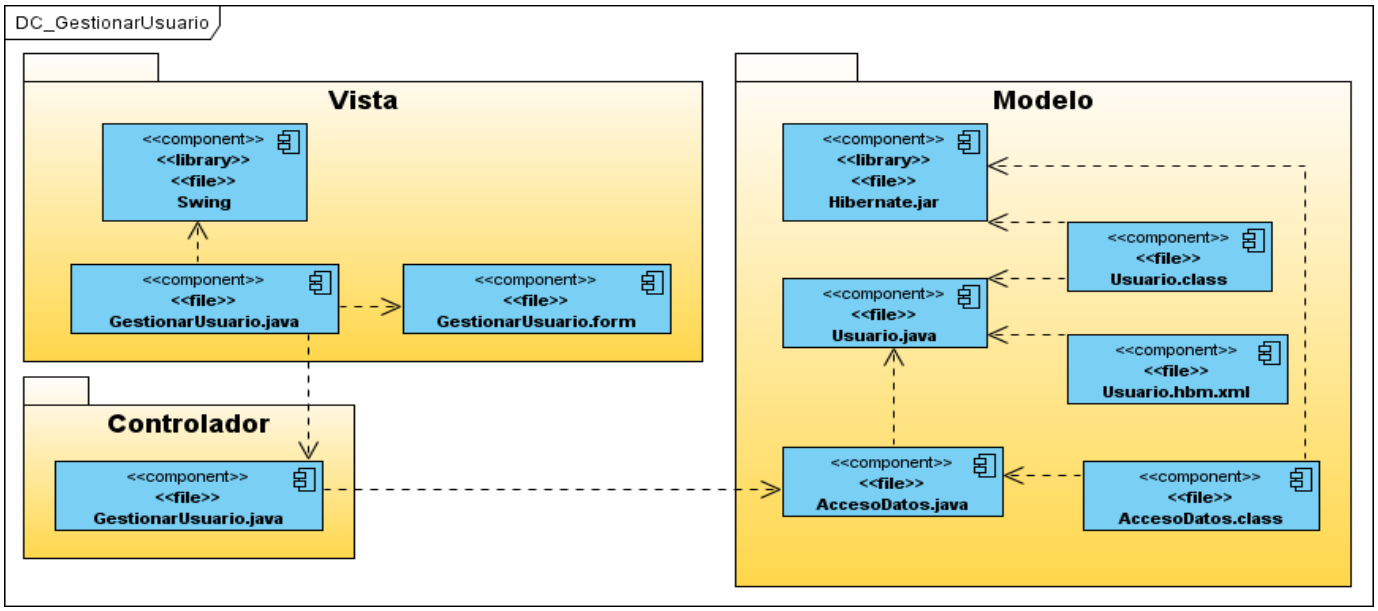
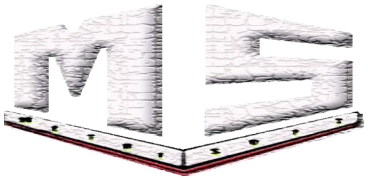


Figura 9: DC_GestionarUsuario.



CAPITULO 4

4.9 Prueba del sistema propuesto

La prueba del *software* se realiza con el objetivo de descubrir errores, teniendo en cuenta como objetivo principal: “Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, creando los procedimientos de prueba que especifican cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables, para automatizar las pruebas.” **(Jacobson, y otros, 2000)**

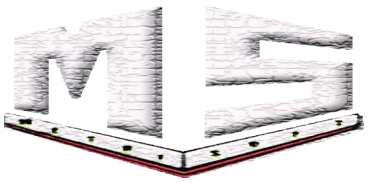
Los casos de prueba diseñados están enmarcados en probar los CU del Sistema críticos y secundarios que a su vez son los escenarios de las pruebas a realizar. Estos casos de prueba encierran el resultado de la interacción de los actores con el sistema y del cumplimiento de las especificaciones del CU como tal.

En los anexos del documento se encuentra un ejemplo de caso de prueba realizado a la aplicación. El software no tiene que ser 100% libre de errores, pero entre más se pruebe el software, mayor cantidad de errores serán encontrados.

A la herramienta desarrollada se le aplicaron pruebas desde inicios de su ciclo de vida por lo que los resultados finales de la técnica de caja negra empleada antes del despliegue, resultaron satisfactorias para los implementadores pues no hubo que reprogramar las funcionalidades - no significa que no haya ningún error-, satisfaciendo además las necesidades de los clientes cumpliendo con los requerimientos establecidos.

4.10 Conclusiones parciales

Después de lo anterior expuesto se puede resumir entonces que el cumplimiento de los aspectos de la construcción de la propuesta de solución permitió establecer el diseño de la herramienta con el diagrama de clases del diseño correspondiente, además de las funcionalidades a implementar seguido de las generalidades de la implementación. Finalmente se realizó el proceso de pruebas a la herramienta implementada basada en la técnica de Caja Negra.



CONCLUSIONES GENERALES

Conclusiones Generales

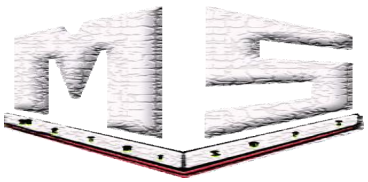
Luego de definir, modelar y construir la herramienta de *software* resultante del presente Trabajo de Diploma, dado por las necesidades establecidas en los estándares de calidad y las Metodologías de Desarrollo del *Software* y cumpliendo con los objetivos predefinidos para con los proyectos productivos de la Facultad 9, se concluye entonces que:

- El estudio realizado para la investigación permitió conocer la realidad existente acerca de la aplicación de los procesos de medición en los proyectos productivos de la Facultad 9, principalmente del empleo de las métricas sobre el producto. Enmarcando en las diferencias existentes entre lo real de la utilización de las métricas y los procedimientos de medición establecidos en las diferentes MDS y el modelo de calidad establecido en la UCI.

- Lo expuesto en la situación problemática y el argumento del objeto de estudio mostraron la relevancia del tema central del presente Trabajo de Diploma y la importancia del análisis y uso de las métricas sobre el producto para evaluar la calidad del *software* permitiendo saber cuánto ha avanzado un proyecto en desarrollo y en qué debe mejorar.

- El estudio y selección de las diferentes tecnologías y herramientas de desarrollo (MDS- RUP *Ultra Light*, CASE-Visual *Paradigm*, Lenguajes de programación- *Java* y modelado- UML, SGBD- *PostgreSQL*, etc.) permitió profundizar en las características de la solución propuesta. Escogiéndose las mejores propuestas de cada categoría de acuerdo a las necesidades existentes y las metas trazadas.

- Según lo establecido en la MDS (RUP *Ultra Light*) la descripción del sistema de la aplicación propuesta permitió capturar los requerimientos funcionales y no funcionales especificados; identificar los actores del sistema, los CU del Sistema y sus respectivas descripciones de textuales; dando paso al diseño de la arquitectura de *software*, estableciendo el Modelo Vista Controlador como patrón arquitectónico de la herramienta, haciendo posible un rápido Diseño de la misma.



CONCLUSIONES GENERALES

- La exposición de las vistas arquitectónicas hicieron posible diagramar cómo quedaría el funcionamiento del sistema en las etapas de Diseño e Implementación del mismo.

- Finalmente se efectuaron las Pruebas al sistema basadas en la técnica de Caja Negra permitiendo validar la funcionalidad de la herramienta de *software* resultante antes de realizar acciones correspondientes al Despliegue de la misma.

Recomendaciones

Se recomienda para el futuro desarrollo del presente trabajo:

- Aplicar la herramienta de *software* resultante del Trabajo de Diploma a los proyectos productivos de la Facultad 9, y extenderlo a toda la comunidad universitaria.

- Proporcionar cursos de capacitación para aquellos desarrolladores que desempeñen el rol de Administrador de la Calidad y Responsable de la Medición dentro los diferentes proyectos productivos, donde se haga referencia a todo lo expuesto en el presente Trabajo de Diploma, con el fin de crear una cultura de control y evaluación de la calidad (medir) en la Facultad 9.

- Desarrollar otra versión en su ciclo completo de la aplicación realizada con las métricas restantes, establecidas por la dirección de calidad de la UCI, pues solo se hace referencia a las métricas del producto, faltando aún las categorías de las métricas del proyecto y del proceso.

Bibliografía

1. Alvarez, J. (2007). *Controles y Métricas Técnicas del Software.*

2. Canós, J. H., Letelier, P., & Penadés, M. d. (2005). *Metodologías Ágiles en el Desarrollo de Software.* Valencia: DSIC_ Universidad Politécnica de Valencia.

3. Cataldi, Z., & Lage, F. (2003). ITBA[en linea].

4. Chrissis, M. B., Konrad, M., & Shrum, S. (2009). *CMMI: Guía para la integración de procesos y la mejora de productos.*

5. Antonio, Angelica de. 2006. GESTIÓN, CONTROL Y GARANTÍA DE LA CALIDAD DEL SOFTWARE. Madrid : Universidad Politécnica de Madrid, 2006.

6. Davis, C. (1993). *Industrial Acceptance of software quality assurance satandards.* IEEE Journal.

7. Dolado, J. (2000). *Medición para la gestión en Ingeniería del Softwre.*

8. Feingenbaum, A. (1961). *Total Quality Control.* Washintongton: McGraw Hil.

9. Ferreira, M., Garcia, F., Ruiz, F., & Bertoa, M. F. (2006). *Medición del Software. Ontología y Metamodelo.* España.

10. Guerrero, H. C. (2007). [En línea] http://hanzcocchi.net/proceso-desarrollo-software-orientado-objetos/.

11. Ishikawa, K. (1998). *¿Qué es el Control Total de la Calidad?: Modalidad Japonesa. .*

BIBLIOGRAFÍA

12. Kulpa, M. K., & Kent, A. (2008). *Interpreting the CCMI. A process improvement approach.* Auerbach Publications.Taylor & Francis Group.

13. Larman, C. (1999). *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* s.l.:P.Hall, Editor.

14. López Requena, M. L. (2006). *Microsoft Solutions Framework.* Málaga.

15. Martin, M., Bertoa, B., Valecillo, A., & Orsina, L. (2002). *Hacia un enfoque.*

16. Nadereau, Y. (2009, junio 23). Propuesta de automatización para el cálculo de las Métricas de Calidad generadas en los proyectos productivos de la Facultad 9. La Habana, La Habana, Cuba.

17. Pérez, O. (2007). *Métricas, Estimación y Planificación en Proyectos de Software.* Universidad de Guadalajara.

18. Pressman, R. (2002). *Ingeniería de software. Un enfoque práctico.* s.l. : MCGraw Hill.

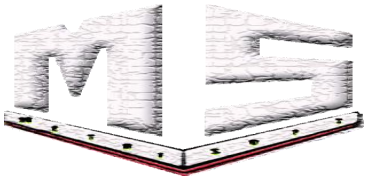
19. Pressman, R. (1993). *Ingeniería de software.Un enfoque práctico.*

20. Pressman, R. S. (2002). *Ingeniería de Software. Un enfoque práctico.*

21. Madrid: Concepción Fernandez Madrid: MCGraw Hill.

22. Rodríguez Verdecia, L., & Lugo García, J. A. (2009, mayo). Diagrama por categorías. La Habana: CALISOFT.

23. Ruilova Rojas, M. E. (2008). Métricas del Producto para el Software (Ingeniería de software Enfoque. *Informe ejecutivo* .



BIBLIOGRAFÍA

24. RUP. (2003). Rational Unified Process.

25. SEI. (2002). *Capability Maturity Model Integration (CMMISM).* Carnegie Mellon University.

26. Suma, P. (2008). *Metodología de desarrollo de software.*

27. Vega Lebrún, C., Rivera Prieto, L. S., & Garcia Santillán, A. (2008). *Mejores prácticas para el establecimiento y aseguramiento de la calidad de software.* unidad multidisciplinaria: CIET.

28. Burbeck, S. (1997, marzo 4). *Aplication programming in Smaltalk-80: How to use Model-View-Controler(MVC).* University of Illions in Urbana- Champaign: Smaltalk Archive.

29. Garlan, D., & Allen, R. (1996). *The Wright Architectural Description Language.* Carnegie Mellon University: Technical Report.

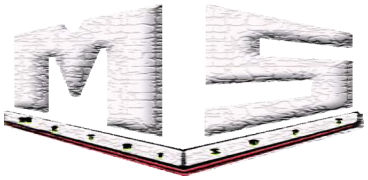
30. Reynoso, C. (2004). *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* s.l.: UNIVERSIDAD DE BUENOS AIRES.

31. Shaw, D. G., & Mary. (1994). *An introduction to software architecture.*

32. Teruel, A. (2000, Septiembre 15). *Arquitectura de capas.* Retrieved abril 21, 2010, from Arquitectura de capas: http://www.ldc.usb.ve/~teruel/ci3715/clases/arqCapas.html

33. Gracia, J. (2005, mayo 27). *Patrones de diseño.* Retrieved abril 21, 2010, from Diseño de Software Orientado a Objetos: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

34. Piñeiro, Y. (2009). *Modelo de Despliegue v1.0.* Universidad de las Ciencias Informáticas, Grupo de Desarrollo de Sistemas de Información Geográfica. Plataforma Soberana LiberGis, Cuba.



BIBLIOGRAFÍA

35. Jacobson, Ivar, Booch, Grady and Rumbaugh, James. 2000. El proceso unificado del desarrollo de software. Madrid : Pearson Educacion, 2000.

36. Schmuller, Joseph. 2000. Aprendiendo UML en 24 Horas. México : Pearson Educacion, 2000.