UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 9

Propuesta de automatización para el cálculo de las Métricas de Calidad generadas en los proyectos productivos de la Facultad 9.

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

AUTORA
Yisel Nadereau Sagarra

TUTOR
Ing. Enrique Pérez Rodríguez

Ciudad de la Habana. 23 de Junio de 2009
"Año del 50 Aniversario de la Revolución."

"La aplicación de la SQA… se puede resumir en una sola frase: Utilizar el tiempo para centrarse en cosas que realmente interesan, pero primero asegurarse que se entiende que es lo que realmente interesa".

Roger S. Pressman

Agradezco:

A mi madre por ser mi universo

A mis abuelos y bisabuelos Xiomara, Dolores, Roberto, Manolo y Argelia

A David por tanto amor y apoyo

A mi tío Ramón por la incondicionalidad

A mi tutor, cotutora y tribunal

A Yisell, Liana, Iván, Denise y Yorjenys mis compañeros de tantos momentos

A Abelardo por su cariño

A mi tío William

A mi madrina Milagros y familia por la luz

A Raiza por cuidar de mi madre

A los radicales del 9105 por un gran comienzo

Al 9406 por el cambio

A Basulto y Yaimara por ser mis brazos derechos

A Radio Ciudad Digital por acogerme en sus brazos y sacar lo mejor de mí

A toda mi gente del Movimiento Código y Letra

A Miguel, Rislaidy, Damaris y Alexis

AGRADECIMIENTOS

A Agustín

A Denlis por estar

A Arturo y a la Negra

A Dick Owen por la grandeza

A Dubiel por la dedicación y esfuerzo

A Yurisbel y a Proenza por ser eternos estudiantes

A Osvaldo por ser mi presente

A la UCI por ser casa y escuela

A la Revolución por crear la UCI y más

A los que vieron aún siendo ciegos

A los que odiaron pudiendo amar

A mi familia, por encima de todas las cosas

Yisel Nadereau Sagarra

DEDICATORIA

A mi madre, mi abuela y mi bisabuela: 

Maricela, Xiomara y Dolores.

A mis soles: 

Christian y César Eloy

RESUMEN

La constante evolución de los métodos para producir software proliferan enormes demandas en las diferentes ramas de la economía y la sociedad, surgiendo la imperante necesidad de garantizar software de calidad. La Universidad de las Ciencias Informáticas (UCI) como eje fundamental de la producción de software en el país sobreviene en el estudio y aplicación de métricas del software, entre ellas las métricas de calidad, como respuesta a dicha necesidad. Pero la comunidad informática todavía no es consciente del uso de las mismas para el cumplimiento de los procedimientos de medición.

De ahí que para lograr que se controlen y evalúen los diferentes atributos (calidad, funcionalidad, fiabilidad, facilidad de uso, eficiencia, eficacia, etc.) que permiten: 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 de diploma 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 de calidad 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

Software, Métricas, Medición, Automatización.

Tablas

Figuras

ÍNDICE

ÍNDICE

ÍNDICE

Introducción

Uno de los aspectos con mayor polémica en el campo de la Informática es la calidad de software. Desde la década del setenta, la calidad de un producto de software ha sido motivo de preocupación para especialistas, ingenieros, investigadores y comercializadores de software, los cuales han realizado gran cantidad de investigaciones al respecto con dos objetivos fundamentales: ¿Cómo obtener un software con calidad? ¿Cómo evaluar la calidad del software? Sorprendentemente, “los factores que definían la calidad del software en el año 1970 son los mismos factores que continúan defendiendo la calidad del software en la primera década de este siglo”; **(Pressman, 2002)** aún conociéndose que cada año los desarrolladores producen aplicaciones y sistemas con tecnologías cada vez más seguras y cambiantes.

Para 1996, Cuba da sus primeros pasos en impulsar el uso y desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC). Comienza así una nueva era en el desarrollo tecnológico y económico del país: la creación del software. Se presentaron demoras en la entrega de los productos de software ya que no cumplían con las necesidades y expectativas del cliente, debido a la ausencia de estándares o medidas que permitieran evaluar la calidad de los mismos dando inicio a una crisis del software que se agudizaba paulatinamente por la limitación de recursos existente.

Actualmente las grandes compañías del orbe aclaman software cada vez más complejos en menos tiempo y menor costo. El aumento del desarrollo de software es considerable, presentando tendencias a realizarlo de forma desorganizada y poco documentado; por lo que se hace necesario el uso de estándares y metodologías que permitan uniformar la filosofía de trabajo en aras de lograr una mayor productividad tanto para la labor de desarrollo como para el control de la calidad. “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". **(Álvarez, 2007)**

Una métrica, según Fenton, es "una asignación de un valor a un atributo de una entidad de software, ya sea un producto o un proceso". **(Fenton, 1991)**. Las métricas se basan en técnicas de mediciones que permiten arrojar resultados cuantitativos durante el ciclo de vida del proceso de desarrollo del software. Estos resultados son obtenidos a partir de cálculos manuales y maniobras de apreciación que impiden de cierta forma su uso periódicamente por parte de los desarrolladores.

Cada año el volumen de información relacionado con las métricas de calidad es cada vez mayor por lo 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." **(Martín, 2004)**. Como respuesta a dicha atenuante surgen las herramientas que gestionan, controlan y calculan estas métricas.

Debido al poco tiempo de existencia de la Universidad de las Ciencias Informáticas (UCI) se constata que su campo productivo cuenta con pocas herramientas a cargo de la medición sistemática y del cálculo de las métricas de calidad en los diferentes procesos de desarrollo del software. De ahí que la presente propuesta surge como necesidad de la dirección del Grupo de Calidad de la Facultad 9 de la UCI de hacer un software que responda a las expectativas de dicha facultad ante la latente situación entorno al desarrollo del software en sus proyectos productivos, los cuales se ven afectados debido a una inadecuada definición de las métricas. Conste afirmar que las pocas métricas malamente utilizadas en ellos no están enfocadas a los objetivos establecidos, a su visión o a su necesidad. En otros casos ni siquiera están definidas. No existen procedimientos para la recopilación de los datos asociados a las mismas, no se realiza la medición de manera periódica y la ausencia de registros históricos de las mediciones actúa en contra de una toma de decisiones oportuna por parte de los directivos. En consecuencia el **problema a resolver** se expone de la siguiente manera:

INTRODUCCIÓN

¿Cómo lograr que se calculen las métricas de calidad generadas en los proyectos productivos de la Facultad 9?

El **objeto de estudio** lo comprenden las Métricas de Calidad en el proceso de desarrollo de software.

Derivándose como **campo de acción** la Automatización del proceso de recopilación de información y generación de métricas de calidad en los proyectos productivos de de la Facultad 9.

El **objetivo general** que se persigue es Implementar una herramienta de software que permita automatizar el cálculo de las métricas de calidad generadas en los proyectos productivos de la Facultad 9.

Y los **objetivos específicos** son:

- Caracterizar los diferentes procesos de desarrollo de herramientas de software que calculen, controlen o generen métricas de calidad.
- Identificar las métricas de calidad adecuadas a utilizar en los proyectos productivos de la Facultad 9.
- Seleccionar las métricas de calidad necesarias a utilizar en la propuesta de solución.
- Modelar y desarrollar una herramienta de software que calcule las métricas de calidad generadas en los proyectos de la Facultad 9.

Con el afán de demostrar como **hipótesis de trabajo** que la implementación de una herramienta de software que automatice el cálculo de las métricas de calidad generadas en los proyectos productivos de la Facultad 9 permitirá medir cuantitativamente los atributos (efectividad, funcionalidad, fiabilidad, tiempo de respuesta, seguridad, confiabilidad, calidad, eficacia, mantenibilidad, eficiencia, usabilidad, integridad, corrección, etc.) de cada proyecto.

Para solucionar la situación problemática y dar cumplimiento a los objetivos trazados se establecieron las siguientes **tareas de la investigación**:

- Análisis exhaustivo de las tendencias actuales del empleo de las métricas de calidad en los diferentes procesos de desarrollo de software y en el aseguramiento de la calidad de los mismos.
- Identificación de los procesos de desarrollo de varias herramientas que controlen, generen y calculen métricas de calidad.
- Identificación de las métricas utilizadas en los proyectos productivos de los polos de la Facultad 9.
- Análisis de los procedimientos definidos para la generación de las métricas y recopilación de datos en los proyectos productivos de los polos de la Facultad 9.
- Selección, a partir de las existentes, de posibles métricas de Calidad a utilizarse en los proyectos productivos de la Facultad 9.
- Análisis y selección de las herramientas y tecnologías a utilizar para el desarrollo del sistema.
- Definición y modelación de una herramienta de software que permita automatizar el cálculo de las métricas de calidad en los proyectos productivos de la Facultad 9.
- Diseño de una Base de Datos que soporte el manejo de datos de las funcionalidades del sistema.
- Implementación de la herramienta de software modelada.
- 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** son factibles en el estudio de las características poco observables del objeto de investigación. Dentro de este grupo se utilizan:

- El método Análisis Histórico-Lógico, el cual es utilizado para identificar si existe a niveles nacional e internacional una herramienta que de solución al fenómeno en cuestión. El presente 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 uso de las métricas como estándares en el proceso de medición de la calidad de procesos de desarrollo del software.

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

- Entrevistas, las cuales son aplicadas a los líderes de los proyectos productivos de la Facultad 9 para reunir la mayor cantidad de información posible sobre los procedimientos de medición, las métricas que se utilizan en sus respectivos proyectos y la recopilación de datos de las mismas.

- Encuesta aplicada a los jefes de polos de la facultad 9 con el objetivo de conocer qué prácticas se realizan en los proyectos que atienden para lograr el aseguramiento y control de la calidad en los mismos. Además de investigar los modelos de calidad, estándares o normas que emplean para sustentar lo anterior.

En vías de que la investigación se desarrolle sobre bases creíbles y contundentes se enfoca el trabajo en la técnica de muestreo No probabilística: Muestreo Intencional, la

cual es la más aconsejada a utilizar porque permite agrupar a los individuos que mayor cantidad de información van a proporcionar.

En el proceso de aplicación de las encuestas se utilizó una:

Población: Todos los Jefes de Polos productivos de la Facultad 9. (Total: 4)

Muestra: Cuatro Jefes de polos productivos de la Facultad 9.

Unidad de estudio: Un jefe de polo productivo de la Facultad 9.

De igual 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: 20)

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

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

La estructura del trabajo de diploma está compuesta por cuatro capítulos:

El Capítulo 1 contempla los conceptos teóricos asociados al dominio del problema seguido de la selección de las métricas de calidad a utilizar en la propuesta junto a la descripción actual del dominio del problema y una ardua argumentación del objeto de estudio. Se plantean las principales dificultades adjudicadas al poco hábito de medir y usar las métricas de calidad durante el desarrollo de los proyectos productivos en la situación problemática, y se incluye un exhaustivo análisis de posibles soluciones existentes sobre la actividad que se investiga.

Se suma el Capítulo 2 mostrando un recorrido por las tendencias actuales de las tecnologías enmarcándose inicialmente en el desarrollo técnico de estas. A partir de varias comparaciones realizadas entre las diferentes tecnologías (metodologías, Sistemas Gestores de Base de Datos (SGDB), lenguajes de modelado y de

programación) comúnmente utilizadas en la producción de software, se seleccionan las herramientas que permitirán desarrollar la propuesta de solución.

Por su parte el Capítulo 3 abarca el comienzo de la etapa ingenieril de la propuesta en cuestión reflejando el cumplimiento de los pasos de la metodología de desarrollo del software (MDS) empleada. Como muestra de ello se exponen los principios de la Captura de Requisitos y las características fundamentales del sistema, las cuales se evidencian en una selección exhaustiva de los Requerimientos Funcionales y No funcionales, Actores del sistema, Casos de Uso del sistema y la documentación de los mismos.

Con gran relevancia continúa el Capítulo 4, el cual contiene todos los aspectos relacionados a la Arquitectura, Diseño, Creación de la Base de Datos y Generalidades de la Implementación durante la construcción de la propuesta de solución, seguido de las características de las pruebas realizadas a la aplicación ya terminada en la fase de Integración de los requerimientos.

Finalmente se unen a lo anteriormente expuesto: las Conclusiones, Recomendaciones, Bibliografía, Referencias Bibliográficas, Anexos y el Glosario de Términos.

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN

Introducción

El desarrollo de la tecnología ha permitido que las personas viabilicen la manera de realizar sus tareas, lo cual no ha facilitado la forma de evaluar y controlar la calidad en los procesos de desarrollo manual, industrial o de software. Todo lo que represente evolución y desarrollo en el campo de la producción debe validarse bajo estándares de calidad. La existencia del ser humano y la lucha constante por su supervivencia dependen en gran medida de una mejor calidad de vida.

La calidad como factor competitivo, es esencial para el comercio internacional, reduce las pérdidas producidas por la no-calidad, alimenta la fidelización de los clientes e incrementa beneficios.

El presente capítulo contempla conceptos teóricos asociados al dominio del problema. En él se seleccionan algunas de las métricas de calidad a utilizar en la propuesta junto a la argumentación del objeto de estudio antes mencionado. Se plantean las principales dificultades adjudicadas al poco hábito de medir y usar las métricas de calidad durante el desarrollo de los proyectos productivos en la situación problemática, así como el análisis de las soluciones existentes de la actividad que se investiga.

1.1 Conceptos asociados al dominio del problema

Como soporte a lo expresado anteriormente se realizará un breve estudio sobre algunos conceptos ligados directamente al dominio del problema. Entre los más relevantes se

seleccionaron los términos de Calidad, Calidad de software, Aseguramiento de la Calidad de software (SQA), Control de la calidad de software, Normas ó estándares de la calidad de software, importancia de la medición en el software, las métricas en el proceso de desarrollo de software, las Métricas de calidad en específico, y otros.

1.1.1 ¿Qué es la calidad?

Antes de la década del cuarenta la calidad era un proceso desarrollado a través del método Inspección/detección de errores aplicado generalmente al trabajo artesanal. Esta **primera etapa** enmarcada desde la Revolución Industrial hasta la Segunda Guerra Mundial, se caracteriza por el surgimiento de nuevas ansias de recuperar el mercado competitivo a través del control estadístico de la calidad; marcando grandes pautas en la lucha continua por evitar que el producto defectuoso llegase a manos del cliente. Aquí el concepto de calidad va asociado al control de las características del producto final. Esta etapa finaliza con un concepto de calidad dado por la conformidad del producto final con las especificaciones iniciales.

La **segunda etapa** abarca el aseguramiento de la calidad con sus raíces en los años cuarenta hasta la década de los setenta, período en el cual se produce el milagro japonés y se difunde el Modelo Deming, en donde la calidad es asociada a la satisfacción de la demanda del cliente externo e interno. Japón marcaría un nuevo mito en la búsqueda de soluciones a un legendario tópico: el control de la Calidad. Deming, Ishikawa, Juran y Crosby serían los iniciadores del gran reto.

El año 1980 avizoraba buenas nuevas con la creación de los términos planificación y medida de la calidad. De ahí el surgimiento de los Modelos de calidad: (1987) Malcolm Bridge (basado en el premio nacional de EE.UU), (1987) ISO 9000, (1992) Premio Europeo a la calidad, etc. Esta **tercera etapa** entraría a jugar su papel en los últimos años del siglo XX y principios del XXI con la Gestión de la Calidad Total. Aquí la calidad implica la participación y la corresponsabilidad de todos los miembros de la organización.

No se pretende diferenciar el concepto de calidad durante las tres etapas mencionadas anteriormente, sino resaltar las diferentes tendencias y el marco histórico en los cuales estuvo y continúa inmiscuido. Es una manera de demostrar los diferentes criterios que han sido incorporados al término durante décadas.

Hoy se conoce que calidad no es sólo evitar la venta de productos porque son de baja calidad y que las mejoras del producto no sólo dependen de prevenir los errores determinando las fuentes de ese error. La alternativa es incorporar la calidad a todas las fases del proceso e implicar a todos los profesionales que intervienen en ellos, con el fin de mejorar los procesos día a día. Esto conduce a hablar de Calidad Total.

El concepto de calidad ha evolucionado a través de los términos Inspección, Control de la Calidad hasta Calidad Total (el más actual). Conste que calidad y control de la calidad no se conciben actualmente como la misma cosa, pero sí están estrechamente involucrados.

Feigenbaum, a quien debemos 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)**

(Crosby, 1970), otro autor norteamericano, considera que la principal "barrera a la calidad" reside en llegar a cambiar las actitudes de algunos operarios incrédulos y en alterar la cultura de la propia organización basada en el miedo, para orientarla hacia la prevención del error y lograr "hacer las cosas bien a la primera".

Ishikawa apuesta por las herramientas de no interrumpir la cadena proveedor-cliente, impulsar la formación continuada, los métodos estadísticos y fomentar la comunicación. **(Ishikawa, 1986)**

Para Deming resultan fundamentales el compromiso de mejora constante y la idea de sustituir la inspección (o control) como forma de conseguir la calidad por una

metodología que implique la participación de todos, rompiendo barreras y fomentando estilos de liderazgo participativos. **(Deming, 1989)**

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)**

Todos estos autores han influenciado de forma directa y excepcional en el desarrollo y evolución del concepto actual de calidad. De ahí que la autora afirma que la calidad es el proceso indispensable para el control y evaluación 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. Tiene carácter global y no debe ser subestimado ni obviado. Está presente en todas las ramas de la economía y la sociedad.

1.1.2 ¿Qué es la calidad de software?

En el campo de la informática el concepto de calidad de software se desenvuelve en un amplio espectro, por lo que existen enésimas formas de concebirla.

Pressman define la calidad de software como la "concordancia del software con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente". **(Pressman, 2002)**

El concepto anterior permite conocer bajo qué circunstancias específicas se incurre en falta de calidad de software:

1. Cuando el software no concuerda con los requisitos establecidos.

2. Cuando no se siguen los criterios definidos en los estándares establecidos como una guía de buenas prácticas en el momento de aplicar la ingeniería de software.

3. Cuando la credibilidad de la calidad del software se expone a duda por el desajuste del mismo para con los requisitos implícitos. Estos requisitos a menudo no son mencionados.

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)**

La norma ISO 8402 define la calidad como la totalidad de características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas. **(Cataldi, y otros, 2003)**

Los aspectos anteriores se muestran al lector para que adquiera habilidades en el momento de interpretar las diferentes afirmaciones expuestas sobre el término calidad, dada la amplitud y carácter global del mismo.

Por otra parte, el hecho de que una empresa o institución presente certificación en calidad de software no garantiza que su producto sea de calidad. Tenga en cuenta que no se certifica la calidad de software, sino los procedimientos necesarios para construir un software con calidad.

En efecto la calidad de software no es más que el conjunto de especificaciones y estándares que deben cumplirse en la producción del software durante todo su ciclo de vida, permitiendo evaluar, controlar y medir la factibilidad, flexibilidad y capacidad del software, siempre y cuando el mismo sea desarrollado por profesionales. De ahí la importancia de: (1) mantener los productos bajo un constante control, (2) desarrollar

mediciones que den como resultado sistemas de alta calidad y (3) contar con métodos y herramientas efectivas que evalúan la calidad con objetividad, no con subjetividad.

1.1.3 Aseguramiento de la Calidad del software (SQA)

El SQA es la etapa de desarrollo de la calidad donde el objetivo primordial es la coordinación de los procesos de desarrollo que llevan al producto. Es decir, que si se coordinan los procesos de desarrollo y fabricación de un producto puede, en cierta manera, garantizarse su calidad. Estudiosos del tema identifican el término de aseguramiento de la calidad con garantía de la calidad, pero se deben tomar las precauciones pertinentes para no confundir el término garantía con el de garantía comercial o de productos.

En un amplio espectro se puede identificar a "la garantía de calidad del software SQA como un "patrón de acciones planificado y sistemático" **(Shoonmaker, 1997)** que se requiere para asegurar la calidad del software". **(Pressman, 2002)**

De ahí que se le conozca como un conjunto de acciones planificadas y sistemáticas necesarias para proporcionar la confianza adecuada de que un producto o servicio satisfará los requerimientos de calidad dados.

El aseguramiento de la calidad vio sus primeros reflejos en paralelo a la calidad en la creación de hardware. Durante los años setenta la aparición de normas ó estándares de de SQA cambiaría la visión con respecto a las causas de las malas prácticas de la calidad en el mundo. Muestra de ello data del año 1974, cuando surge la especificación militar norteamericana MIL-S-52779 titulada "Programa de requerimientos para el aseguramiento de la calidad del software". Conocida actualmente como la primera norma de SQA.

En la década de los ochenta se inicia el aseguramiento de la calidad en la industria del software mediante el uso de estándares en sistemas de calidad para los procesos. El establecimiento de nuevas o conocidas políticas de calidad contribuye en gran medida a

lograr la calidad de software pero no la aseguran. Se hace necesario para lograr el SQA el control o evaluación de la calidad de software.

"La garantía de calidad consiste en la auditoría y las funciones de información de la gestión. El objetivo de la garantía de calidad es proporcionar la gestión para informar de los datos necesarios sobre la calidad del producto, por lo que se va adquiriendo una visión más profunda y segura de que la calidad del producto está cumpliendo sus objetivos. (…) si los datos proporcionados mediante la garantía de calidad identifican problemas, es responsabilidad de la gestión afrontar los problemas y aplicar los recursos necesarios para resolver aspectos de calidad". **(Pressman, 2002)**

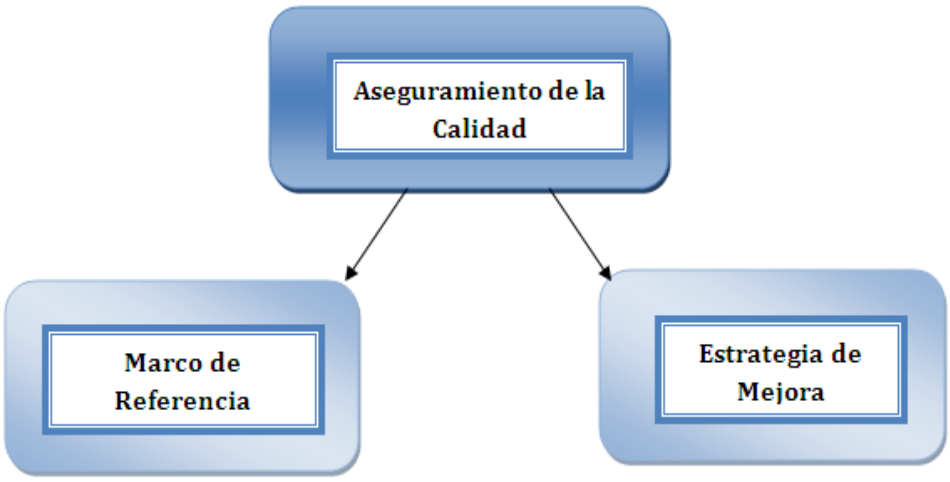


Figura 1: Estrategias de Trabajo del SAQ

La planificación del SQA para cada aplicación debe realizarse antes de que la aplicación comience a ser desarrollada y puede constatarse su presencia durante el proceso de desarrollo de software a través de las siguientes actividades de SQA:

- Métricas de software para el control del proyecto.
- Verificación y validación del software a lo largo del ciclo de vida (Incluye las pruebas y los procesos de revisión e inspección).

- Gestión de configuración del software.

Estas actividades se ejecutan a través del uso de métodos de SQA tales como: (revisiones técnicas y de gestión, inspecciones, pruebas y auditorías). Una de las actividades de SQA mayormente beneficiosa la comprenden los cambios, sin embargo, representan una amenaza para la calidad del proceso de desarrollo de software. Para contrarrestar lo anterior algunos estudiosos del tema aconsejan realizar un proceso de control de cambios acorde a la metodología de desarrollo de software que se esté utilizando o se decida a utilizar.

Generalmente los proyectos de desarrollo de software cuentan con un equipo de SQA encargado de las actividades de aseguramiento de la calidad necesarias. Las reglas creadas en el seno del equipo SQA ayudan en gran medida a que el equipo de desarrollo del proyecto de software obtenga un producto final con calidad. De ahí que se pueda afirmar que el aseguramiento de la calidad es un proceso de control. El equipo enfoca su trabajo desde el punto de vista del usuario centrando su trabajo en dar respuestas a interrogantes tales como: (1) ¿Se está realizando el proceso de desarrollo de software de acuerdo a los estándares establecidos? (2) ¿Las disciplinas técnicas están desarrollando su papel como es debido?

Cuando escuche hablar de Aseguramiento de la calidad de software puede pensar inmediatamente en la estrategia que aplican los responsables de SQA en los equipos de desarrollo de proyectos de software con el fin de garantizar la calidad de los productos generados en el proyecto y del proceso utilizado. Para asegurar la calidad debe revisar la calidad de los entregables de planificación del proyecto y los entregables de valoración del proyecto. Se revisa además, el nivel de apego al modelo de proceso de desarrollo del software y a los planes de Verificación, Gestión de Proyecto y Gestión de la Calidad, documentando las desviaciones encontradas. Se resume pues, en todas aquellas acciones que se realizan con el fin de proporcionar confianza en que el producto cumple con determinadas reglamentaciones de calidad, agolpando (1) su

completitud en la verificación de que estos requisitos cumplan con las expectativas del cliente y (2) su efectividad con la realización periódica de auditorías y la evaluación continua de aquellos factores que afecten la calidad del producto.

1.1.4 ¿Qué es el Control de la Calidad de software?

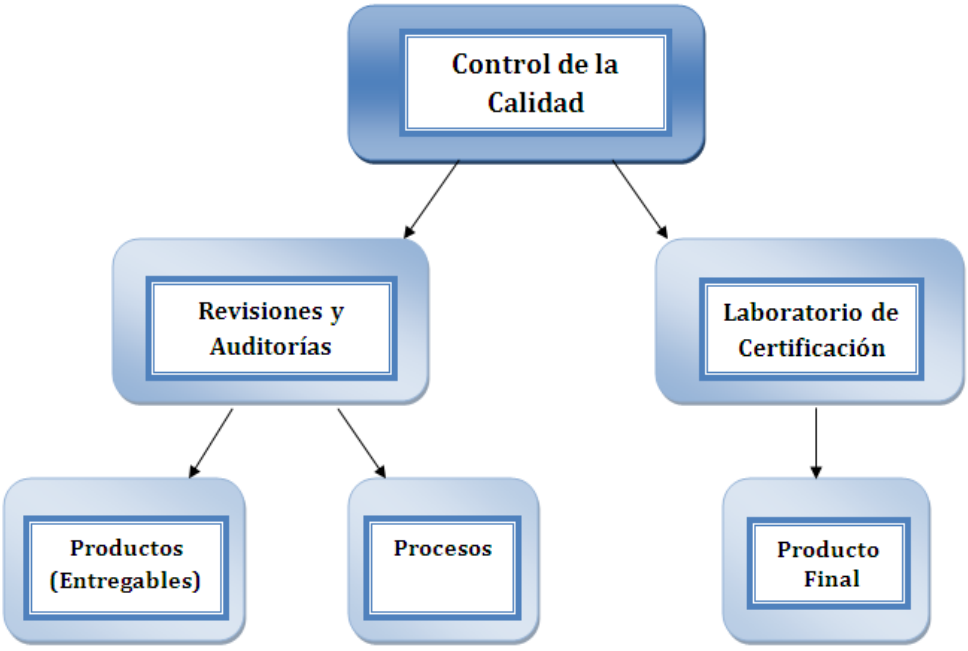


Figura 2: Estrategias de trabajo que sigue el Control de la Calidad

A grandes rasgos nos referimos a control de la calidad cuando lo que interesa es comprobar la conformidad del producto con respecto a las especificaciones de diseño del mismo. El objetivo de las acciones de control de la calidad consiste en identificar las causas de la variabilidad para establecer métodos de corrección y de prevención y para lograr que los productos fabricados respondan a las especificaciones de diseño. Sin embargo el control de la calidad de software centra sus propósitos más allá de la conformidad del producto para con las especificaciones del diseño.

Un producto de software para ser explotado durante un largo período, necesita ser confiable, mantenible y flexible para disminuir costos de mantenimiento y

perfeccionamiento durante el tiempo de explotación. Controlar la calidad del software precisa definir parámetros, indicadores o criterios de medición.

Los ingenieros del software aplican mediciones e indicadores que le permitan evaluar cuantitativamente la calidad de las diferentes actividades en los flujos de trabajo que se han establecido al aplicar la ingeniería del software. Para obtener esta evaluación de calidad, el ingeniero debe utilizar medidas técnicas, que evalúan la calidad con objetividad, no con subjetividad. **(González, 2001)**

El control de la calidad de software conforma una de las principales vértebras para llevar a feliz término las buenas prácticas para el SQA. Representa una estrategia para asegurar el mejoramiento continuo de la calidad y la satisfacción de los clientes internos ó externos permitiendo que las actividades de un proceso de desarrollo de software ocurran según fueron planeadas.

A lo expuesto anteriormente se suma Pressman quien aboga que “el control de la calidad es una serie de inspecciones, revisiones y pruebas utilizados a lo largo del proceso del software para asegurar que cada producto cumple con los requisitos que le han sido asignados (…) incluye un bucle de retroalimentación (feedback) del proceso que creó el producto (…) que “es esencial para reducir los defectos producidos”. Las actividades de control de la calidad pueden ser manuales, completamente automáticas o una combinación de herramientas automáticas e interacción humana”. **(Pressman, 2002)**

Por lo que se resume que el control de la calidad no es más que las diferentes actividades operativas que nos permiten comprobar si un producto cumple con los requerimientos de calidad establecidos. Se debe realizar una comparación de los requisitos y el producto finalmente desarrollado, para garantizar que el mismo cumpla con las especificaciones del cliente. El uso de técnicas estadísticas en su proceder lo identifican bajo estrictas circunstancias, como Control Estadístico de la Calidad.

1.1.5 Normas o estándares de Calidad de software

Las normas o estándares rigen el comportamiento de las personas. Surgen como una necesidad de documentar toda actividad que discipline y organice el quehacer productivo de un grupo de individuos. Conforman una guía para las prácticas de calidad y productividad. Nos proveen los medios para que todos los procesos se realicen siempre de la misma forma ante el impedimento de mejorarlos.

Un estándar es el documento aprobado por consenso por un organismo reconocido, que proporciona reglas, pautas y/o características para uso común, con el objeto de obtener un óptimo nivel de resultados en un contexto dado. **(ISO/IEC, 1991)**.

Según la ISO, se denomina estándar a "un conjunto de acuerdos documentados que contienen especificaciones técnicas u otros criterios precisos para ser usados constantemente, como reglas, lineamientos o definiciones de características. Todo esto con la finalidad de asegurar que los materiales, productos, procesos y servicios son óptimos para su propósito". **(Álvarez, 2004)**

En la industria del software se emplean varios estándares de calidad ISO que a continuación se nombran:

ISO 9001: Describe el sistema de calidad utilizado para mantener el desarrollo de un producto que implique diseño.

El estándar ISO 9001 es uno de los más populares en términos de usabilidad. Más de cien países lo han adoptado para su uso convirtiéndolo en el principal medio a través del cual los clientes juzgan el trabajo de un desarrollador de software. No obstante presenta limitantes con respecto a la generalización de sus especificaciones al estar creado para la Industria pero no especifica cuál (tipo de industria).

ISO 9000-3: Documento específico que interpreta el ISO 9001 para el desarrollador de software.

ISO 9004-2: Proporciona las directrices para el servicio de facilidades del software como soporte de usuarios.

ISO 9126: Ha sido desarrollado en un intento de identificar los atributos clave de calidad para el software. Portabilidad: la facilidad con que el software puede ser llevado de un entorno a otro. **(Tecnomaestros)**

ISO/IEC 9126

Es un estándar internacional para la evaluación del software enfocado en la calidad del producto. Consta de las siguientes partes:

- **9126-1** Modelo de calidad (Parte 1)
- **9126-2** Métricas Externas (Parte 2)
- **9126-3** Métricas Internas (Parte 3)
- **9126-4** Calidad en el uso (Parte 4)

Este estándar proviene del modelo establecido en 1977 por McCall y sus colegas, que propusieron un modelo para especificar la calidad del software. A pesar de que es uno de los más antiguos se ha extendido en todo el mundo. **(Mojena, 2007)**

En síntesis un estándar o norma es parte esencial de un sistema de calidad como proceso indispensable en la gestión de la calidad. Conforma un conjunto de políticas a seguir para garantizar el feliz término del desarrollo de un producto.

1.1.5.1 Algunas entidades a cargo de la Estandarización

Muchas son las entidades a relieves nacional e internacional que dedican sus esfuerzos a la creación de normas y estándares para medir la calidad. Grandes organizaciones e industrias dependen en gran medida de la existencia de antiguos y nuevos estándares

para combatir la variación de la calidad de los productos y establecerla bajo políticas uniformes y sólidas. El éxito y confianza de lo que puedan producir estas empresas, qué hacer y cómo hacerlo, requiere ineludiblemente de la presencia de estándares de calidad para la producción del software y otros. Debido a ello sectores de la economía y la sociedad han dado luz verde a instituciones para la normalización para que desarrollen tan elocuente trabajo. A continuación se procede a realizar una breve reseña sobre las entidades de estandarización que se consideran más importantes a nivel internacional:

ISO: Organización Internacional para la Estandarización o International Organization for Standardization. Organismo internacional no gubernamental cuya sede está situada en Ginebra y enfoca sus tareas en desarrollar la normalización con carácter mundial. Establece normas internacionales, industriales y comerciales conocidas como “normas ISO”.

IEEE: Instituto de Ingenieros Eléctricos y Electrónicos o The Institute of Electrical and Electronics Engineers. Organización radicada en Estados Unidos de gran prestigio a nivel mundial con carácter técnico-profesional.

SEI: Instituto de Ingeniería de Software o Software Engineering Institute. Instituto federal estadounidense fundado por el Congreso de los Estados Unidos para desarrollar e investigar modelos de evaluación y mejora en el desarrollo de software. Es financiado por el Departamento de Defensa de los Estados Unidos y administrado por la Universidad Carnegie Mellon. Presenta excelentes referencias en Ingeniería de Software por desarrollar en 1991 el modelo SW-CMM, el cual ha marcado el punto de partida a la vasta gama de modelos asociados al concepto de capacidad y madurez. Su versión actual: CMMI.

1.1.6 Modelos de Calidad

Los modelos de calidad conforman un conjunto de buenas prácticas para el ciclo de vida de un proceso de desarrollo de software enfocado generalmente en procesos de gestión. Según la metodología que se emplee y los objetivos del negocio un modelo de calidad explica QUÉ hacer y no CÓMO hacer.

1.1.6.1 CMM

CMM ó SW-CMM: Modelo de Capacidad y Madurez o Capability Madurity Model Software. Primera definición de Modelo de Capacidad y Madurez desarrollada por el SEI. Conjunto de prácticas ó procesos agrupados en Áreas de procesos. Para cada una de estas áreas define las buenas prácticas siguientes:

- Definidas en un procedimiento documentado.
- Provistas (la organización) de los medios y formación necesarios.
- Ejecutadas de un modo sistemático, universal y uniforme (institucionalizadas)
- Medidas
- Verificadas

Dichas áreas se agrupan en cinco niveles de madurez (Inicial, Repetible, Definido, Gestionado y Optimizado). Si una organización logra institucionalizar todas las prácticas antes mencionadas en un nivel, puede decirse que ha alcanzado ese nivel de madurez. Actualmente CMM está sustituido por CMMI.

1.1.6.2 CMMI

Denominado como Modelo de Capacidad y Madurez o Capability Maturity Model Integration, CMMI fue desarrollado por el SEI y está orientado a la mejora de procesos en diferentes niveles de madurez hacia proyectos específicos. Es un proceso de mejora que proporciona a las organizaciones los elementos esenciales de procesos eficaces

que se utiliza además, para guiar la mejora del proceso a través de un proyecto, una división, o toda una organización. CMMI ayuda a integrar funciones tradicionalmente separadas de organización, establecer objetivos de mejora de procesos y prioridades, servir de orientación para los procesos de calidad, y proporcionar un punto de referencia para evaluar los procesos actuales.

Entre sus beneficios podemos citar:

- Su suite de productos cuenta con las últimas mejores prácticas para el desarrollo de productos y servicios y mantenimiento.
- Mejora las prácticas de modelos de calidad anteriores con gran versatilidad.
- Permite a las organizaciones: (1) Vincular con mayor explicitud las actividades de Ingeniería y gestión a sus objetivos de negocio, (2) Ampliar el alcance de la visibilidad en el ciclo de vida del producto y las actividades de ingeniería para garantizar el producto o servicio, (3) Cumplir con las expectativas de los clientes, (4) Incorporar las lecciones aprendidas en otros ámbitos de las mejores prácticas (por ejemplo: la medición, gestión de riesgos y gestión de proveedores), (5) Aplicar de forma más robusta las prácticas de alta madurez, (6) Abordar otras funciones de organización fundamental para sus productos y servicios y (7) Cumplir cabalmente con las normas ISO. **(SEI, 2002)**

Lo esencial del empleo de los modelos de calidad involucrados en la familia CMM es que las mediciones deben establecerse de forma alineada con los objetivos del negocio (usados regularmente para justificar indicadores de costo y esfuerzo, proveer beneficios, etc.) para así lograr que las personas practiquen la medición correctamente.

1.1.6.2.1 Medición y Análisis como Área de Procesos de CMMI

Algunas organizaciones realizan actividades de mediciones de forma superficial, lo cual puede arrojar pocos o nulos beneficios. Medición y Análisis es una de las Áreas de

procesos (AP) de CMMI que reenfoca sus esfuerzos en lograr el seguimiento del análisis y colección de mediciones con el fin de evaluar la habilidad de estas para soportar las necesidades de administración de la información de las organizaciones. Su meta principal es establecer que los objetivos de las mediciones deben estar alineados con los objetivos y necesidades de las organizaciones, y que los resultados de las mediciones deben direccionar dichas necesidades.

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, 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, 2008)**

La presente AP provee las especificaciones arrojadas en las revisiones a los documentos mencionados anteriormente, las cuales deben ser redactadas de forma precisa y sin ambigüedades. Explica además cómo se analizarán y cuantificarán dichas mediciones. Sirve de apoyo en la identificación de los procedimientos de medición a ejecutar en los diferentes proyectos productivos de nuestra universidad a partir de la

acogida de CMMI como modelo de calidad por parte de la Dirección de Calidad en la UCI en el curso 2005-2006.

1.1.7 Importancia de la Medición en el Software

A decir de Tom De Marco, "usted no puede controlar lo que no se puede medir". De ahí se puede afirmar que medir es conocimiento, y el mismo permite avanzar sobre bases sólidas.

Habitualmente medimos aspectos que consideramos palpables, sin embargo en la ingeniería de software la medición se aleja de lo común provocando desacuerdos sobre qué medir y cómo evaluar las medidas.

La medición se puede aplicar al proceso del software con el intento de mejorarlo sobre una base continua. Se puede utilizar en el proyecto del software para ayudar en la estimación, el control de calidad, la evaluación de la productividad y el control de proyectos. Finalmente, el ingeniero de software puede utilizar la medición para ayudar a evaluar la calidad de los resultados de trabajos técnicos y para ayudar en la toma de decisiones tácticas a medida que el proyecto evoluciona. **(Pressman, 2002)**

Sepa que durante más de 30 años, esta rama de la Ingeniería de software, ha sido objeto de investigación y estudio. No obstante, actualmente coexisten muchas personas que aún no reconocen la importancia de la medición como parte indispensable de la calidad durante un proceso de desarrollo de software.

Los procesos, los recursos y los productos son entidades importantes a estudiar en el entorno de la realización de medidas aplicadas al desarrollo de programas informáticos. Una vez definidas las entidades, se procede a seleccionar los atributos a medir. A favor del enriquecimiento de una conciencia para hacer uso correcto de la medición Pressman dijo:

Existen cuatro razones para medir los procesos del software, los productos y los recursos:

- **Caracterizar:** Se caracteriza para comprender mejor los procesos, los productos, los recursos y los entornos y para establecer las líneas base para las comparaciones con evaluaciones futuras.

- **Evaluar:** Se evalúa para determinar el estado con respecto al diseño. Las medidas utilizadas permiten conocer cuando los proyectos y los procesos están fuera de control. También se evalúa para determinar el impacto de la tecnología y las mejoras del proceso en los productos y procesos.

- **Predecir:** Se predice para poder planificar. Esto se hace porque se quieren establecer objetivos alcanzables para el coste, planificación, y calidad (de manera que se puedan aplicar los recursos apropiados). Las medidas de predicción también son la base para la extrapolación de tendencias, con lo que la estimación para el coste, tiempo y calidad se puede actualizar basándose en la evidencia actual. Las proyecciones y las estimaciones basadas en datos históricos también ayudan a analizar riesgos y a realizar intercambios diseño/coste.

- **Mejorar:** Se mide para mejorar cuando se recoge la información cuantitativa que ayuda a identificar obstáculos, problemas de raíz, ineficiencias y otras oportunidades para mejorar la calidad del producto y el rendimiento del proceso. **(Pressman, 2002)**.

Alegando además que para ejecutar un proceso de medición correcto se deben realizar las siguientes actividades:

- **Formulación:** Obtención de medidas y métricas del software apropiadas para la representación del software en cuestión.

- **Colección:** Mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
- **Análisis:** Cálculo de las métricas y la aplicación de herramientas matemáticas.
- **Interpretación:** Evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.
- **Retroalimentación:** Recomendaciones obtenidas de la interpretación de métricas técnicas transmitidas al equipo que construye el software. **(Pressman, 2002)**.

La medición proporciona al líder de proyecto la evidencia cuantificable que este necesita como apoyo a la toma de decisiones, basándose en anotaciones realistas y no en la subjetividad, hace más visible el desarrollo y le permite anticiparse a los problemas, ayudando así, a que este pueda mantener el control. Además, estas observaciones le permitirán al líder en caso de atrasos durante el proceso de desarrollo, trazar las estrategias necesarias para cumplir con la planificación. **(Arias, y otros)**

La medición del software se refiere a derivar un valor numérico para algún atributo de un producto de software. El proceso de medición es una actividad clave, ya que de este depende que los resultados puedan ser fiables y fáciles de evaluar. Los pasos a seguir en este proceso son los siguientes: (a) Seleccionar los componentes a evaluar. (b) Medir las características de los componentes con las métricas de software. (c) Identificar las mediciones anómalas. (d) Analizar los componentes anómalos. **(Dávila, y otros)**

Se puede asegurar entonces, que la medición sería en gran medida la clave para la prevención de errores durante el proceso de desarrollo de software. Una toma de decisiones factible por parte de los directivos del proyecto dejaría de ser una utopía,

junto a la trasparencia de los diferentes procesos de recopilación de datos. Pese a todo lo expuesto, se constata la carencia de una cultura necesaria sobre la medición en la producción del software que no permite conocer el grado de madurez que proporciona la misma al producto final.

1.1.8 Métricas en el proceso de desarrollo de software

Uno de los estándares que con mayor fuerza se investigan en la UCI son las métricas. Las mismas están valoradas como unos de los exponentes de solución más fehacientes a los problemas causados por la falta de medición en los diferentes proyectos productivos.

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

Conforman un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo del software y los proyectos de mantenimiento. **(L. Briand, y otros, 1996)**. De ahí que:

"Las métricas del software pueden ser utilizadas para que los profesionales e investigadores tomen las mejores decisiones. Proporcionan información objetiva que contribuye al mejoramiento de los procesos y productos de software lo cual, evidentemente, favorece al logro de la calidad, siempre que se haga un uso adecuado de las mismas". **(Pressman, 2002)**

Y 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, 2002)**

Muñoz las clasifica además en:

- **Directas o Indirectas:**

  - Directa: Una métrica de la cual se pueden realizar mediciones sin depender de ninguna otra métrica y cuya forma de medir es un método de medición.

  - Indirecta: Una métrica cuya forma de medir es una función de cálculo, es decir, las mediciones de dicha métrica utilizan las medidas obtenidas en mediciones de otras métricas directas o indirectas. **(Muñoz, 2006)**

Y otros autores en:

- **Internas o Externas:**

- Interna: Puede ser aplicada a un producto de software no ejecutable (como una especificación o código fuente) durante el diseño y la codificación.

- Externa: Usa medidas de un producto de software, derivadas del comportamiento del mismo, a través de la prueba, operación y observación del software. **(Autores, 2004)**

Existe una gran gama de métricas para el proceso de desarrollo de software, pero no todas conforman un soporte práctico para el desarrollador. Inclusive pueden llegar a causar tedio 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)**

Se realiza entonces un llamado de atención durante el proceso de selección de las métricas a utilizar, porque si bien son necesarias para saber cuantitativamente cuánto se ha avanzado durante el ciclo de vida del proyecto de software; hacer uso indebido de ellas puede retrasarlo.

1.2 Argumentación del Objeto de Estudio: Métricas de Calidad en el proceso de desarrollo de software

Bajo el contexto de un proceso maduro de desarrollo de software los ingenieros de software deben aplicar métodos y herramientas para medir si la alta calidad se está llevando a cabo. "Un buen ingeniero del software utiliza mediciones que evalúan la calidad del análisis y los modelos de diseño, el código fuente, y los casos de prueba que se han creado al aplicar la ingeniería del software". **(Pressman, 2002)**

Estudiosos del tema como McCall y Cavano definieron un grupo de factores de calidad a tener en cuenta para el desarrollo de métricas de calidad del software. Estos factores evalúan al software desde tres puntos de vista distintos:

- Operación del producto (utilizándolo)

- Revisión del producto (cambiándolo)
- Transición del producto (modificándolo para que funcione en un entorno diferente, <<portándolo>>). **(McCall, 1978)**

Los aportes expuestos anteriormente datan de hace más de dos décadas y hoy en día un ingeniero de software puede apostar con sus resultados la veracidad de la calidad del producto que desarrolla.

Para justificar la existencia de las métricas de calidad, "se argumenta que éstas deben ser enunciadas y utilizadas para administrar el proceso de desarrollo y debe ser conforme al producto de software particular". **(Davis, 1993)**. El ingeniero 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.
- Tomar acción remedial si los niveles de las métricas crecen o exceden los niveles objetivos establecidos.
- Establecer metas de mejoras específicas en términos de las mismas métricas. (Vega, Rivera, García, 2008)

Pueden recopilarse enésimas métricas de calidad pero el objetivo fundamental del equipo de desarrollo del proyecto para alcanzar un producto con calidad es medir errores y defectos. Estas variables proveen métricas que proporcionan indicadores referentes a la efectividad de las actividades de aseguramiento y control de la calidad en grupos o particulares.

Gilb sugiere las siguientes medidas de calidad:

- **Corrección:** Un programa debe operar correctamente o proporcionará poco valor a sus usuarios. La corrección es el grado en que el software lleva a cabo su función requerida.

  Medida: Defectos por KLDC

  Donde un defecto se define como una falta verificada de conformidad con los requisitos.

- **Facilidad de Mantenimiento:** Es la facilidad con la que se puede corregir un programa si se encuentra un error, se puede adaptar si su entorno cambia, o mejorar si el cliente desea un cambio de requisitos.

  Medida: Tiempo medio de cambio (TMC)

  Donde TMC es el tiempo que el sistema se tarda en analizar la petición de cambio, en diseñar una modificación adecuada, en implementar el cambio, en probarlo y en distribuir el cambio a todos los usuarios.

- **Integridad:** Este atributo mide la capacidad de un sistema para resistir ataques tanto accidentales como intencionados. Para medir la integridad se deben definir dos nuevos atributos: (1) Amenaza y (2) Seguridad.

  1. **Amenaza:** es la probabilidad (estimable o deducible empíricamente) de que un ataque determinado ocurra en un tiempo determinado.

  2. **Seguridad:** probabilidad (estimable o deducible empíricamente) de que se pueda repeler un ataque determinado.

  Medida: integridad =∑ [(1-amenaza) x (1-seguridad)]

Donde se suman la amenaza y la seguridad para cada tipo de ataque.

- **Facilidad de uso:** Es un intento de cuantificar cuán amigable puede ser el producto de software con el usuario. Se mide en función de cuatro características:

1. Habilidad intelectual y/o física requerida para aprender el sistema.

2. El tiempo requerido para llegar a ser moderadamente eficiente en el uso del sistema.

3. Aumento neto en productividad (enfoques que el sistema reemplaza). Cuando alguien utiliza el sistema moderada ó eficientemente.

4. Valoración subjetiva de la disposición de los usuarios hacia el sistema. Generalmente se obtiene a través de cuestionarios. **(Gilb, 1988)**

Otras métricas de Calidad:

- **Calidad (1)**

Medida: calidad= Errores/ KLDC

Donde:

Errores: indicador que reúne la cantidad de errores encontrados por líneas de código.

KLDC: indicador que reúne la cantidad de miles de líneas de códigos que presenta el producto hasta el momento de la medición.

Muy relacionadas con la definición de densidad de defectos del software se encuentran otras medidas que se agrupan con el nombre de "tasas", las cuales se comportan como indicadores de la calidad del proceso. Entre las medidas más utilizadas se encuentran:

- **Tasa de Propagación de Defectos**

TPD= Número de defectos ocasionados al corregir defectos
Número de defectos corregidos

Donde TPD es la Tasa de propagación de defectos.

- **Tasa de Efectividad de las pruebas**

TEP= Número de objetos probados al menos una vez * 100
Número de objetos totales

Donde TEP es la Tasa de efectividad de las pruebas.

Según Melián y Ballesteros, las ecuaciones anteriores permiten realizar una política de pruebas y captura de medidas muy valiosas para el desarrollo de aplicaciones y su control de calidad. **(Melián, 2003)**

- **Eficacia de la Eliminación de Defectos (EED)**

Medida: EED= E / (E+D)

Donde:

E es el número de errores encontrados ante de la entrega del software.

D es el número de defectos encontrados después de la entrega del software.

Pressman asegura que EDD es "una métrica de calidad que proporciona beneficios tanto a nivel del proyecto como del proceso" que mide "la habilidad de filtrar las actividades de la garantía de la calidad y de control al aplicarse a todas las actividades del marco de trabajo del proceso". **(Pressman, 2002)**.

Y funciona de la siguiente forma:

> Cuando no se han encontrado defectos en el software el valor de EED es 1. En la práctica D siempre será mayor que cero y el valor de EED se aproxima a 1. "A medida que E aumenta, es probable que el valor de D disminuya (los errores se filtran antes de que se conviertan en defectos)." **(Pressman, 2002)**

También se puede emplear la EED para evaluar la habilidad de un equipo en encontrar errores en una actividad antes de pasar a la otra. Si no se encuentran los n errores establecidos de acuerdo a la revisión en la actividad actual, se pasan a la siguiente actividad hasta ser encontrados. En ese contexto EED se vuelve a definir como:

- **Versión ampliada de la Eficacia de la Eliminación de Defectos (EED)**

  Medida: EEDi= Ei / (Ei + Ei+1)

  Donde:

  Ei es el número de errores encontrado durante la actividad de ingeniería de software i.

  Ei+1 es el número de errores encontrado durante la actividad i+1.

Existen otras métricas de calidad muy ligadas a las métricas de puntos de función (PF). En esta dirección los puntos de función son utilizados analógicamente a las LDC (líneas de código) como medidas de calidad. Como precondición se impone el cálculo manual de los puntos de función. **(Ver procedimiento de cálculo en Anexo 1)**.

- **Calidad (2)**

  Medida: calidad= Errores / PF

  Donde:

Errores: indicador que reúne la cantidad de errores encontrados por líneas de código.

PF: indicador resultado del cálculo de la métrica de Punto de Función.

La fiabilidad es un atributo de calidad que nos permite saber que tan libre de fallos está un sistema. De ahí que la métrica a emplear sea la **densidad de defectos** (DD), la cual según Dávila y Mejía se define como la cantidad de errores que puede ocurrir en un lapso de tiempo determinado.

El estándar **ISO 9126-3** recopila en su haber las Métricas Internas de la Calidad del Producto de Software. Según Gonzalo Mena estas métricas:

- Se aplican durante la etapa de desarrollo del software.
- Permiten medir la calidad de los entregables intermedios.
- Permiten predecir la calidad del producto final
- Permiten iniciar acciones correctivas tempranas en el ciclo de desarrollo.

(Mena, 2006).

Entre ellas resaltan:

- Métricas de Funcionalidad: Adecuidad

  Nombre: **Completitud de la Implementación funcional**

  Esta métrica permite conocer que tan completa está la implementación funcional del software que se desarrolla.

Precondiciones: Se debe contar las funciones faltantes detectadas mediante una revisión y compararlas con las cantidad de funciones expresadas según los requerimientos funcionales existentes.

Medida: completitud= 1 – A / B

Donde:

A es el número de funciones faltantes.

B es el número de funciones descritas en la especificación de requerimientos.

Mientras más cerca esté completitud de 1, más completa será la implementación funcional del software. (0 <= completitud >= 1)

- Métricas de Fiabilidad: Madurez

  Nombre: **Suficiencia de las Pruebas**

  La presente métrica permite conocer cuántos de los casos de prueba están respaldados por el Plan de Pruebas.

  Precondiciones: Contar la cantidad de pruebas planeadas y compararlas con el número de pruebas requeridas.

  Medida: suficiencia= CPP / CPR

  Donde:

  CPP es el número de casos de prueba del plan.

  CPR es el número de casos de prueba requeridos.

Mientras mayor sea la variable suficiencia, mejor será la suficiencia de las pruebas.

- Métricas de Eficiencia: Comportamiento

  Nombre: **Tiempo de respuesta**

  Esta métrica permite conocer la eficiencia de las llamadas al sistema operativo (SO) y a la aplicación.

  Precondiciones: Estimar el tiempo de respuesta basado en lo anterior. Puede ser calculado o simulado.

  Medida: Trespuesta= tiempo calculado o simulado

  El Trespuesta debe ser lo más corto posible.

- Métricas de Mantenibilidad: Cambiabilidad

  Nombre: **Registro de Cambios**

  Esta métrica verifica si se registran los cambios a la especificación y a los módulos con comentarios en el código.

  Precondiciones: Debe registrarse la proporción de información sobre los cambios a los módulos.

  Medida: RCambios= CC / FMM

  Donde:

  CC es el número de cambios a funciones o módulos con comentarios confirmados.

FMM es el número de funciones o módulos modificados.

Mientras más cercano este RCambios de 1 se considerará cada vez más registrable. (0 <= RCambios >= 1) El parámetro 0 implica que no existe control de cambio o pocos cambios. **(Mena, 2006)**

1.3 Selección de las Métricas de Calidad a utilizar en la propuesta

Anteriormente se expusieron las métricas de calidad mayormente reconocidas en la medición de los atributos (efectividad, fiabilidad, eficacia, calidad, funcionalidad, etc.) que definen un producto de alta calidad antes de ser entregado a su usuario final. Para la selección de las métricas de calidad idóneas a utilizar en el desarrollo de la propuesta en cuestión se tuvieron en cuenta las métricas presentes en los Planes SQA de cada proyecto productivo de la Facultad 9 siempre y cuando los mismos estuviesen elaborados y correctamente actualizados, además del apoyo en métricas de este tipo ya presentes en fuentes bibliográficas seguras. Aclarar que no se han mencionado todas las métricas existentes sino las que son consideradas óptimas para satisfacer las necesidades identificadas en los proyectos productivos de la facultad.

Para lograr un balance fiable de la selección de dichas métricas se esclarece que hubo algunas que no se consideraron del estándar ISO 9126-3 para la selección. Tal es el caso de las métricas de calidad de Transportabilidad (Conformidad de la Transportabilidad), Entendibilidad (Funciones evidentes) y otras.

La siguiente tabla muestra las métricas seleccionadas para ser utilizadas en la propuesta de solución.

No.	Métricas de Calidad	Selección		Tipo	Observaciones
		Sí	No		
1.	Miles de líneas de código (KLCD)	X		Directa	Responde al atributo de Corrección

2.	Defectos por KLDC	X		Directa	Responde al atributo de Corrección
3.	Tiempo Medio de cambio (TCM)	X		Directa	Responde al atributo de Facilidad de mantenimiento
4.	Integridad	X		Indirecta	Responde al atributo de Integridad
5.	Facilidad de Uso	X		Indirecta	Ver características en el Acápite 1.2 Métricas de Calidad en el proceso de desarrollo de software. Responde al atributo de Usabilidad.
6.	Errores encontrados antes de la entrega del software (E)	X		Directa	Responde al atributo de Efectividad.
7.	Defectos encontrados después de la entrega del software (D)	X		Directa	Responde al atributo de Efectividad.
8.	Eficacia de la eliminación de defectos (EED)	X		Indirecta	Responde al atributo de Eficacia.
8.1	Eficacia de la eliminación de defectos (EEDi)	X		Indirecta	Para evaluar la habilidad de los equipos de un proyecto para hallar errores en las actividades de Ingeniería de software. **(Ver características**

					en Acápite 1.2)
9.	Calidad (1)= E / KLDC	X		Indirecta	Responde al atributo de Calidad.
10.	Calidad (2)= E / PF			Indirecta	Tiene como precondición el cálculo de la métrica de Puntos de función (PF). **(Ver Anexo 1)** Responde al atributo de Calidad.
11.	Tasa de propagación de defectos	X		Indirecta	Responde al atributo de Fiabilidad.
12.	Tasa de efectividad de pruebas	X		Indirecta	Responde al atributo de Fiabilidad.
13.	Completitud de la Implementación funcional completitud= 1 – A / B	X		Indirecta	Responde al atributo de Funcionalidad.
14.	Transportabilidad		X	Indirecta	Responde al atributo de Transportabilidad.
15.	Entendibilidad		X	Indirecta	Responde al atributo de Entendibilidad.
16.	Suficiencia de las Pruebas suficiencia= CPP / CPR	X		Indirecta	Responde al atributo de Suficiencia pero en el área de pruebas.
17.	Tiempo de respuesta	X		Directa	Responde al atributo de Eficiencia.
18.	Registro de Cambios RCambios= CC / FMM	X		Indirecta	Responde al atributo de Mantenibilidad.
19.	Densidad de Defectos (DD)		X	Directa	Responde al atributo

					de Fiabilidad.

Tabla 1: Métricas de calidad seleccionadas para la propuesta de solución

1.4 Descripción actual del dominio del problema

Luego de realizarse un sondeo sobre la situación actual del empleo de las métricas de calidad en la UCI y constatar su efecto a partir de la opinión de especialistas de la Dirección de Calidad UCI se plantea que a través de una estrategia introducida por dicha dirección los equipos de desarrollo de los proyectos productivos de la universidad encuentran una guía que compila las pautas para proceder ante la necesidad de realizar procesos de mediciones utilizando métricas de calidad. Sepa que dicha estrategia está basada en los objetivos estipulados por el Nivel 2 del modelo de calidad: CMMI. No obstante se incurre, de manera generalizada, en el incumplimiento de la estrategia debido a la existencia de factores negativos en el ámbito productivo tales como:

- Persiste la inexistencia de espacios o ambientes donde la comunidad universitaria pueda mantenerse informada sobre la importancia del proceso de medición y el papel de las métricas de calidad en el mismo.

- El descontrol de la información manipulada en virtud del desarrollo de los proyectos entorpece las actividades de documentación al respecto y en consecuencia se afectaría una futura toma de decisiones robusta.

- Al Grupo de Métricas de Calidad de la Dirección Central de Calidad se le dificulta realizar la revisión de los resultados aglomerados en grandes volúmenes de reportes generados en los proyectos productivos a partir del uso de las métricas de calidad.

- Se constata el hecho de que los Líderes de proyectos y los desarrolladores desconocen todo lo relacionado con el uso de métricas para el control de la eficiencia y calidad de los sistemas informáticos.

Lo anteriormente planteado y otros factores que serán ampliados en la situación problemática exacerban la necesidad de implementar una herramienta que permita calcular las métricas de calidad generadas en los proyectos productivos con el fin de revertir esa situación en el entorno productivo de la Facultad 9.

1.5 Situación Problemática

La Universidad de las Ciencias Informáticas (UCI) surge como primera obra de la Batalla de Ideas, institución adjunta al Ministerio de la Informática y las Comunicaciones (MIC) no solo como una entidad educativa sino también productiva que realiza múltiples proyectos destinados a la informatización de la sociedad cubana y la comercialización de software. Uno de sus mayores retos es lograr producir software con calidad. “La calidad requiere de control y a su vez, se hace necesario un método universalmente aceptado que permita diagnosticar en forma ordenada aquello que deseamos resolver o mejorar”. **(Vega, y otros, 2008)**. No se han escatimado esfuerzos en el estudio de normas o estándares, además de la investigación en el centro de las métricas, las cuales permiten evaluar de forma estadística los elementos de la calidad de software.

No obstante, la comunidad científica no ha sido consciente de la importancia de la medición como manera eficaz para evaluar algunos parámetros como funcionalidad, complejidad y eficiencia durante el proceso de producción de software. De ahí que en el desarrollo de los productos no se refleja una organización y planificación adecuadas, los productos son revisados en la etapa final, lo cual trae consigo que el producto no sea entregado en la fecha establecida, por lo que existe una pérdida de tiempo y recursos tanto material como humano. **(Cue, 2007)**

La Facultad 9 como célula de la UCI reúne en su seno 4 polos recién estructurados, que en su corto tiempo de vida y quehacer productivos han dado empobrecidas muestras de garantía de la calidad de software en sus proyectos.

Estos proyectos productivos, en su mayoría, no cuentan con un equipo de SQA, y con un personal encargado del rol afín a la medición, lo cual dificulta el progreso de los diferentes procesos de desarrollo debido a la inexistencia de registros históricos que permitan medir, evaluar y controlar el trabajo realizado; dando vía libre a la creación de variados y complejos proyectos de software que acceden a su etapa de desarrollo sustentándose en bases empíricas. Proyectos de software que a su vez no responden al llamado de solucionar problemas (por ejemplo: Correcciones, Toma de decisiones, Falta de control, Exceso de gasto, Coste de mantenimiento y Evaluación de nuevos métodos) a partir de los procesos de medición que permitirían: (1) Proporcionar requerimientos verificables expresados en términos medibles, (2) Proporcionar evidencia cuantificable para tomar decisiones,(3) Identificar problemas anticipadamente y facilitar la visibilidad del proceso de desarrollo,(4) Producir predicciones de coste y plazos justificables (ajuste de gastos), (5) Identificar módulos críticos y determinar estrategias de pruebas y (6) Valorar efectos en la productividad y la calidad.

Lo anteriormente expuesto se agudiza con la falta de definición de una guía que estructure las buenas prácticas del establecimiento y Aseguramiento de la Calidad de Software en los proyectos productivos de la Facultad 9. De ocurrir lo contrario el nivel de certeza sobre las prácticas a seguir ayudaría a evaluar los procesos de desarrollo y las mismas a su vez, estarían fundamentadas en parámetros objetivos, aportando un paradigma mejor estructurado en los procedimientos actuales de trabajo.

1.6 Análisis de posibles soluciones existentes

El presente acápite centra sus esfuerzos en el análisis de algunas soluciones existentes a niveles nacional e internacional para el Control y Gestión de métricas para el desarrollo del software. Otros ejemplos provienen de nuestro patio universitario dando solución a aspectos colaterales que sirven de apoyo a la actividad que se investiga por lo que se persigue el objetivo de reunir estos aspectos para incrementar el soporte de validez de la solución en proceso.

1.6.1 Sistema de Gestión y Control de Métricas

Herramienta de software desarrollada en la facultad 8 de la UCI para el control y gestión de las métricas del desarrollo de software, la cual no se encuentra vigente en los proyectos productivos de la Facultad 9 además de no factibilizar el cálculo de las métricas de calidad que generan los mismos.

1.6.2 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")**.

1.7 Conclusiones Parciales

Luego de analizar los conceptos teóricos antes expuestos puede asegurarse la esencia del papel de las métricas en los procedimientos que miden, evalúan y controlan el desarrollo del proceso de construcción del software. Se reafirma además, el protagonismo de las medidas de los atributos específicos del proyecto, del proceso y del producto para calcular las métricas de calidad con el fin de proporcionar indicadores que guíen acciones de gestión siempre y cuando se utilicen las métricas adecuadas.

Se concluye que la medición como rama de la ingeniería de software y actividad rectora del empleo de las métricas requiere de cambios en la cultura de las organizaciones creadoras de software de ahí que se proceda al empleo de una estricta selección de métricas a partir de la recopilación de datos y el cálculo y evaluación de las mismas. En este caso métricas de calidad a favor de las especificaciones de la investigación.

CAPÍTULO 2

TENDENCIAS Y TECNOLOGÍAS ACTUALES A DESARROLLAR

Introducción

La tecnología permite en gran medida que el hombre vaya avanzando a partir de un rumbo marcado de acciones racionales. La misma debe ser evaluada a partir de atributos tales como: su utilidad y eficacia práctica. Su proceso evolutivo tiene sus antecedentes en la necesidad que presente cierto individuo de resolver cierto conflicto. De ahí que el papel de la tecnología en el desarrollo del software es indispensable.

En este capítulo el lector podrá realizar un recorrido por las tendencias actuales de la tecnología enmarcándose inicialmente en el análisis de algunas herramientas para el desarrollo del software. A partir de varias comparaciones realizadas entre las diferentes tecnologías (por ejemplo: metodologías de desarrollo de software (MDS), sistemas gestores de base de datos (SGDB), lenguajes de modelado y lenguajes de programación, etc.) comúnmente utilizadas en la producción de software, se seleccionan las herramientas que permitirán desarrollar la propuesta de solución en cuestión.

2.1 Metodologías de desarrollo de software

"Para lograr una producción de un software exitosa es preciso emplear tecnologías y metodologías de desarrollo de software que permitan desarrollar, instalar y mantener un producto de este tipo. 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, 2002)**

Con la evolución continua de las aplicaciones informáticas se ha observado un aumento significativo en la creación de metodologías a favor de lograr productos con calidad y dar soporte al ciclo de desarrollo del proceso, pero no se ha logrado una metodología de carácter universal.

La Universidad de las Ciencias Informáticas como institución productora de software deviene en el uso de varias metodologías con el fin de mejorar la calidad de sus productos durante el proceso de desarrollo de los mismos.

2.1.1 Metodologías robustas o pesadas

Las metodologías tradicionales se centran especialmente en el control del proceso. Las mismas establecen rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán durante el proceso de desarrollo.

Dentro de este grupo se puede mencionar entre las más significativas:

Rational Unified Process (RUP)

Microsoft Solution Framework (MSF)

2.1.1.1 Rational Unified Process (RUP)

El Proceso Unificado de Software es una metodología de desarrollo creada con el objetivo de producir software con calidad. Es un proceso desarrollado por la Corporación "Rational Software" bajo la tutela de Philippe Kruchten e Ivar Jacobson. Actualmente conforma una división de IBM. Se destaca por ser: (1) dirigido por casos de uso, (2) iterativo e incremental y (3) centrado en la arquitectura. Iterativo porque se producen varios ciclos en su desarrollo en los cuales se concreta más el producto como

resultado del ciclo anterior. E incremental porque en cada ciclo se realizan mejoras a favor de que el producto resultante se adecue cada vez más a las necesidades de los clientes.

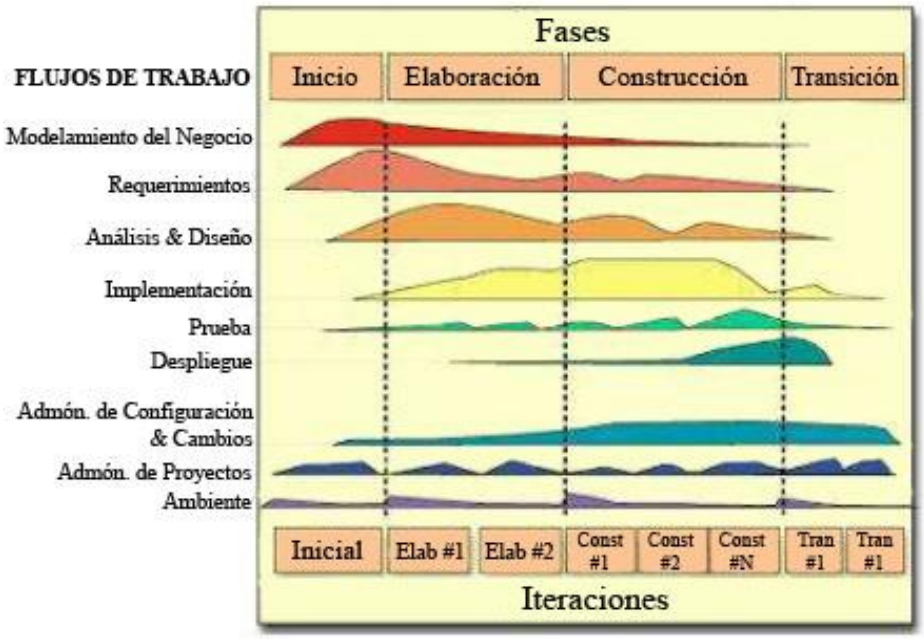


Figura 3: Fases e iteraciones del ciclo de vida de RUP

2.1.1.2 Microsoft Solution Framework (MSF)

Como bien su nombre lo constata esta metodología de software trabaja sobre el campo de Microsoft y obliga al equipo de trabajo a utilizar herramientas de su proveedor. Es una metodología flexible que se aplica a proyectos grandes y pequeños pero con grandes volúmenes de documentación por lo cual el equipo de desarrollo debe ser paciente para lograr un producto bajo los requerimientos estipulados. Divide el proceso de desarrollo en cinco fases:

- **Visión:** donde se hace una descripción general de las metas y restricciones del proyecto. Aquí se identifican las tareas y entregables.

- **Planeamiento:** los miembros del equipo de trabajo y el cliente definen el qué y el cómo de la solución a implementar.

- **Desarrollo:** en esta fase los miembros del equipo de trabajo desarrollan y prueban la solución a implementar, este punto comprende el desarrollo de código y los entregables de documentación resultantes.

- **Estabilización:** es en la que los miembros del equipo de trabajo y del cliente prueban la solución completa, estabilizando la misma en función de los desvíos encontrados y preparando la misma para su liberación en producción.

- Implementación: donde el equipo implementa la solución tecnológica y sus componentes, estabiliza la implementación, transfiere el proyecto a producción y soporte, y obtiene la aprobación final del cliente sobre el proyecto. **(Mendoza, 2004)**

MSF se caracteriza por ser: (1) Adaptable (2) Escalable (3) Flexible y (4) Tecnológicamente Agnóstica.

2.1.2 Metodologías ágiles

Las metodologías ágiles surgen tras una reunión celebrada en Utah-EEUU en febrero de 2001 donde nace el término “ágil” aplicado al desarrollo del software. En esta reunión participaron un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y responder a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

Las mismas están orientadas específicamente para proyectos pequeños y se centran en otras dimensiones (por ejemplo: el factor humano, el producto, etc.) Ofrecen mayor

valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Son muy efectivas en proyectos con requisitos cambiantes y en caso de la exigencia de reducir drásticamente el tiempo de desarrollo se mantienen los principios que garantizan la alta calidad.

Valga resaltar la presencia de exponentes ágiles como:

Extreme Programming (XP)

RUP Ágil

RUP Ultra Light

2.1.2.1 eXtreme Programming (XP)

XP es una metodología de desarrollo del software ágil creada por Kent Beck en 1996 para rescatar el proyecto del Sistema exhaustivo de compensaciones de Chrysler (C3). "Su objetivo principal está en lograr la completa satisfacción del cliente, proporcionándole lo que necesita y cuando lo necesita, además de potenciar al máximo el trabajo en grupo que solo puede funcionar con programadores experimentados en la materia, y solo para pequeños grupos de estos". **(Mendoza, 2004)**

Tiene como requisitos que: (1) debe ser utilizada en proyectos que consten de un pequeño equipo de trabajo, (2) debe utilizarse además en proyectos con plazos de entrega como máximo de 3 meses, y (3) que el usuario final o cliente forme parte del equipo de trabajo.

Esta metodología se basa en tres características fundamentales:

Pruebas Unitarias: representan un conjunto de pruebas dirigidas a los principales procesos. Las mismas avizoran posibles fallos que pudieran ocurrir en el futuro. **Refabricación:** la reutilización de código es la base de sus fundamentos. Para ello se crean patrones o modelos estándares flexibles al cambio.

Programación en Pares: no es más que la integración o participación de dos desarrolladores en un proyecto en la misma estación de trabajo. **(Mendoza, 2004)**

2.1.2.2 RUP Ágil

Para las personas vinculadas al campo de la ingeniería de software no es un secreto que RUP se puede utilizar al estilo de cascada tradicional ó ágil, sólo que lo anterior depende de la forma en que cada cual lo adapte al ambiente del proyecto de desarrollo en el cual trabaja. RUP Ágil se divide en dos variantes:

- **Agile Unified Process (AUP):** versión simplificada de RUP que describe fácilmente el aseguramiento del negocio de aplicaciones de software usando técnicas y conceptos ágiles. Divide el proceso de desarrollo en cuatro fases al igual que RUP: Inicio, Elaboración, Construcción y Transición.

- **Enterprise Unified Process (EUP):** es la versión extendida de RUP Ágil. Si se está familiarizado con el entorno de RUP podrá constatar que EUP brinda dos nuevas fases: Producción y Retiro.

2.1.2.3 RUP Ultra Light

Debido a que RUP es un proceso muy general y grande, antes de usarlo es necesario configurarlo y adaptarlo a las características de la institución que lo utilice. Muchas personas se han dedicado a tomar los elementos más significativos y genéricos del mismo para generar versiones reducidas, que cumplan las expectativas para la construcción de un producto software. Surge así la metodología conocida como RUP Ultra Light, la cual 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.

Para desarrollar la MDS RUP Ultra Light se deben seguir los siguientes pasos:

1. **Realizar un Diagrama de Casos de Uso:** Una vez identificadas todas las funcionalidades conque debe contar el software que se pretende construir, se debe representar todos estos casos de uso en un diagrama teniendo en cuenta además, a los actores involucrados y las relaciones existentes entre ellos (actor-caso de uso, actor- actor, caso de uso- caso de uso).

2. **Priorizar los Casos de Uso a trabajar:** Tras haberse identificado todos los casos de uso conque debe contar el sistema, se debe hacer una lista de prioridad de casos de uso, donde la prioridad es el riesgo que conllevan. Los riesgos es todo lo que puede afectar el buen desarrollo del sistema como puede ser: necesidad de cambiar la arquitectura, no escoger los requisitos adecuados, no construir un sistema correcto, etc. Se debe determinar cuáles son los requerimientos más importantes a desarrollar en las primeras iteraciones y cuáles deben dejarse para más tarde.

3. **Generar los Documentos de Caso de Uso:** Para un mejor entendimiento de la funcionalidad asociada a cada caso de uso se necesita un documento que describa de forma breve o extendida lo que hace el caso de uso, más allá de la representación gráfica de su Diagrama de Casos de Uso,

   El documento de Caso de Uso debe ser generado por el Analista del proyecto y debe tener de una forma u otra 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:** Un diagrama de secuencia permite conocer la forma en la que los objetos se comunicarán en una pantalla para cumplir su objetivo a través de la ordenación temporal de mensajes durante su ciclo de vida. No es indispensable hacer este gráfico pero ayudará a que el Arquitecto de Software comprenda mejor lo que debe hacer, y los Implementadores podrán hacer mejor su trabajo.

5. **Diseñar el Framework del proyecto:** El Arquitecto de Software del proyecto diseñará las clases que serán usadas en todo el Software. Es un trabajo bastante delicado ya que el mal diseño de las clases involucra que no sean implementadas las funcionalidades de cada clase de forma correcta, lo cual conllevará a escribir un *"Spaghetti Code"*, que significa que el código estará difuso. Esta etapa de diseño debe ser revisada cuidadosamente y se recomienda la utilización de Patrones de Diseño de Software de ser posible.

6. **Creación de la Base de Datos:** El diagrama de clases del diseño desarrollado constituye el punto de partida para el diseño de la Base de Datos del proyecto pues se puede utilizar la Capa de Datos del mismo, donde se alojan las clases Entidad.

7. **Construir la máscara la WebSite o Aplicación Windows:** Simultáneamente al desarrollo de los pasos 4, 5 y 6 se pueden ir realizando las plantillas para la

creación de las páginas web ayudándose de los gráficos de las GUI (Graphic User Interface o Interfaz Gráfica de Usuario) que se encuentran en los Documentos de Casos de Uso.

8. **Programar las funcionalidades de los Casos de Uso:** Una vez terminadas las clases, se comienzan a programar las funcionalidades de los Casos de Uso. Para ello, los programadores se apoyan en los documentos de CU desarrollados por los analistas y basándose en el Diagrama de Secuencia y en las clases diseñadas por el Arquitecto escriben el código que se necesita 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 CU.

10. **Integrar los requerimientos concluidos**: Al finalizar es indispensable unir lo que se ha realizado por diferentes programadores de forma tal que el sistema funcione como un todo y ponerlo a disposición de los usuarios.

Se deben repetir de los pasos 3 al 10 por cada iteración que se haya programado para el proyecto, para de esta forma poder controlarlo.

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

Tras analizar las características de las metodologías mencionadas anteriormente resalta la presencia de ciertas particularidades que permiten reconocer el papel de cada una de ellas en el proceso de desarrollo de software.

Teniendo en cuenta dichas particularidades se procede a seleccionar a RUP Ultra Light como la metodología que apoya el desarrollo de la propuesta, debido a que se adapta a

las condiciones de trabajo existentes. En este caso el equipo de desarrollo está compuesto por una sola persona lo cual implica que la misma desempeñe todos los roles de Ingeniería de Software necesarios para el buen desarrollo de la propuesta de solución en un período corto de tiempo. De ahí que se haga necesario reducir el proceso de desarrollo de la herramienta en cuestión apostando a que dicha metodología responda a las necesidades establecidas por el cliente y permita comenzar la modelación a partir de lo que se quiere y lo que el sistema debe hacer (la captura de requisitos) además, de agilizar la interpretación de los procesos en versiones simplificadas.

Se insiste en aclarar que con el fin de aplicar los conceptos ágiles a la propuesta de solución actual no se pretende en lo absoluto minimizar la importancia de la documentación durante el proceso de desarrollo del software.

2.3 Selección de las herramientas de desarrollo.

Consecuente con la problemática planteada se propone desarrollar una Aplicación Desktop o 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 de calidad generadas en dichos proyectos. Una herramienta sencilla y factible que responda a las necesidades de los equipos 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.

Este tipo de aplicación no es más que una solución completa de interfaz gráfica de usuario que ofrece iconos, barras de herramientas, programas e integración entre aplicaciones con habilidades para ofrecer al usuario una interacción amigable. Cada entorno de escritorio se distingue por su aspecto y comportamiento particulares. Entre sus ventajas podemos encontrar que el usuario está más acostumbrado a su uso y manejo, no se necesita servidor y que como aplicación pesada debe correr en el lado del cliente.

La aplicación Desktop a desarrollar como parte de la propuesta de solución tiene como característica inicial cumplir con los términos “libre” y “multiplataforma” de acuerdo a los estándares establecidos y las facilidades que proporcionan dichos aspectos al desarrollador y usuarios involucrados.

Para poder seleccionar las herramientas a emplear en el desarrollo de la aplicación Desktop se realizará un estudio de las principales tecnologías mayormente utilizadas en ámbito productivo de la UCI: lenguajes de programación y de modelado, además de algunos sistemas gestores de bases de datos.

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

UML representa el lenguaje gráfico de modelado más utilizado en el campo de la producción del software. Diagrama la realidad de un requerimiento determinado y tiene como complemento la programación orientada a objetos.

“El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas”. **(Larman Craig, 2006)**.

Aunque UML no especifica que metodología utilizar, sí especifica métodos o procesos. De ahí que juega un papel fundamental en el modelado de la propuesta de solución en cuestión permitiendo al cliente, a través de diagramas, entender con mayor exactitud el proceso de desarrollo de software a realizar. Estos diagramas se dividen en tres grupos: (Diagramas de Estructura, Diagramas de Comportamiento y Diagramas de Interacción).

Llevando a cabo el uso de UML como lenguaje de modelado en el proceso de desarrollo de la propuesta de solución se cumple en efecto con uno de los Lineamientos Mínimos

de Calidad (LMC) establecido por la Dirección de Calidad para la producción del software en la UCI.

2.5 Visual Paradigm: Herramienta CASE de la propuesta de solución

Para modelar los diagramas propios del desarrollo de la aplicación propuesta se utilizará la herramienta Visual Paradigm Enterprise 3.4, herramienta case libre que permite el modelado de varios lenguajes de programación y otras tecnologías de forma fácil y asequible al desarrollador.

"Visual Paradigm es una herramienta CASE (Ingeniería de Software Asistida por Ordenador, en inglés Computer Aided Software Engineering) muy potente que utiliza UML como lenguaje de modelado. Soporta el ciclo de vida completo de desarrollo de un software, desde la fase de análisis hasta el despliegue del mismo. Permite realizar ingeniería directa o inversa sobre el software, es capaz a partir de un modelo relacional en diferentes SGBD de desplegar todas las clases asociadas a las tablas. Soporta múltiples usuarios trabajando sobre el mismo proyecto. Se caracteriza por su robustez, usabilidad y portabilidad". (Novo, Marrero, 2008)

En un aparte se comenta que existen varias herramientas CASE para estos fines. Tal es el caso del Rational Rose Enterprise, la más conocida en el entorno productivo-educativo de la Facultad 9. Es el producto más completo de la familia Rational Rose. Perfecto para el modelado UML y permite "trabajar en diseños de base de datos con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos". **(GSInnova)**. Lamentablemente es un producto de carácter propietario.

2.6 Principales lenguajes de programación.

En la actualidad se puede contar con una variada gama de lenguajes de programación que comprenden la vía de comunicación entre un usuario y su ordenador. Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las

instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático.

Los lenguajes de programación se clasifican en: (1) Lenguaje Máquina y (2) Lenguajes de programación de bajo y alto niveles.

En vías de seleccionar el lenguaje de programación ideal para implementar la aplicación propuesta se consultan a continuación los lenguajes mayormente utilizados en el entorno productivo de la UCI para la creación de aplicaciones Desktop.

2.6.1 C++

A mediados de la década de los ochenta, Bjarne Stroustrup desarrolló C++, el cual es una mejora del lenguaje C y cuyas características fundamentales son su capacidad de programación orientada a objetos (POO) y la manipulación de los mismos. Desde la óptica de los lenguajes orientados a objetos, C++ es un lenguaje híbrido.

Tiene como particularidades la sobrecarga de operadores, la creación de nuevos tipos que se comportan como tipos fundamentales, herencia múltiple, la creación de clases abstractas y otras.

Presenta 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.

2.6.2 CSharp (C#)

European Computer Manufacturers Association (ECMA) y la ISO aprobaron a C# como un estándar años después de ser creado por Microsoft Corporation en el 2001 como parte de la plataforma .NET (interfaz de programación de aplicaciones). C# es un lenguaje de alto nivel y orientado a objetos. Su base sintáctica está basada en el uso del modelo de objetos de .NET (muy similar a Java) y se deriva de C y C++. Pese a

que C# forma parte de la plataforma .NET, es un lenguaje de programación independiente diseñado para crear programas sobre dicha plataforma.

2.6.3 Java

Todo proceso eficiente debe contar con la ayuda de herramientas de apoyo al desarrollo que puedan integrarse con los entornos de desarrollo conocidos. Tal es el caso de Java, diseñado en 1990 por James Gosling, de Sun Microsystems, como software para dispositivos electrónicos de consumo y solución simultánea a todos los problemas a los cuales se exponen los desarrolladores por la proliferación de arquitecturas incompatibles.

En el período de noviembre de 2006 a mayo del 2007 Sun Microsystems liberó la mayoría de las tecnologías Java bajo la GNU Licencia General Pública o GNU GPL por lo que actualmente todo el paquete Java es libre aunque la biblioteca de clases para ejecutar los programas no lo es aún.

Java reduce en un cincuenta por ciento los errores más comunes de programación de lenguajes como C y C++ al eliminar muchas de las características de éstos.

Soporta además las tres características propias del paradigma de la orientación a objetos (OO): encapsulación, herencia y polimorfismo; e incorpora funcionalidades inexistentes en C++ como por ejemplo: la resolución dinámica de métodos.

Se sintetiza entonces, que 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.

2.7 Fundamentación de la selección del lenguaje de programación a utilizar en la implementación de la propuesta de solución.

Después de haber realizado un exhaustivo análisis de algunos de los lenguajes de programación mayormente utilizados en el entorno productivo de la UCI y la Facultad 9, se procede a realizar la selección del lenguaje idóneo para desarrollar la aplicación Desktop propuesta.

Los lenguajes mencionados con anterioridad presentan entre sí particularidades y mejoras que a su vez los diferencian el uno del otro. Estas mejoras forman parte indisoluble del proceso evolutivo llevado a cabo por desarrolladores o simples usuarios de ordenadores que no escatiman esfuerzos en crear nuevos lenguajes cada vez más fáciles, eficientes y seguros.

Dicho esto se escoge al lenguaje de programación Java para desarrollar la aplicación Desktop debido a que las aplicaciones de Java resaltan por su extrema seguridad ya que no acceden a zonas delicadas de memoria o de sistema. Java implementa además, un método ultraseguro de autentificación por clave pública para evitar que los crackers desarrollen efectivas vías de intrusión.

El IDE NetBeans GUI integrado con su constructor aglutina las tecnologías de Java Desktop, lo que permite construir fácilmente una aplicación de escritorio. La tecnología Java de escritorio se puede utilizar para crear aplicaciones cliente y applets que son rápidos, seguros y portátiles. Estas últimas son algunas de las características a tener en cuenta para cumplir con las expectativas del cliente.

Argumentar además, que este lenguaje permite cumplir con los principios establecidos en la política de producción de la administración de la Facultad 9 de crear productos de software multiplataforma y libres que respondan al llamado internacional de eliminar las restricciones del software privativo. Un gran reto a las puertas del siglo XXI.

2.8 Sistemas Gestores de Base de Datos (SGBD).

Para el correcto manejo y acceso eficiente a los datos durante el proceso de construcción de la herramienta se necesita de un SGBD que la soporte. Un SGDB “es

un software específico que permite a los usuarios crear, mantener y manipular la base de datos". **(Ves, 2006)**. Un sistema gestor de base de datos permite además la reducción del tiempo de desarrollo de las aplicaciones, la administración, seguridad e integridad de los datos, hace posible el acceso concurrente y optimiza la recuperación del sistema ante la presencia de fallos.

En consecuencia se procede a realizar brevemente un estudio de los SGBD más reconocidos a nivel internacional y usualmente utilizados en las labores productivas de la Facultad 9 de la UCI.

2.8.1 MySQL

MySQL ha demostrado que puede competir con los grandes nombres del mundo de la gestión de bases de datos. En la actualidad es una solución viable y de misión crítica para la administración de datos que incorpora muchas de las funciones necesarias para otros entornos y conserva su gran velocidad. Por esta razón muchos usuarios lo reconocen como el SGBD Open Source (código fuente abierto) más popular de Internet.

MySQL es un SGBD relacional, multihilo y multiusuario que es muy utilizado en plataformas (Linux/Windows-Apache-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación de datos.

2.8.2 PostgreSQL

PostgreSQL es un sistema de gestión de base de datos relacional y orientado a objetos que se distribuye bajo la licencia de software libre BSD ó Distribución de Software Berkeley (en inglés, Berkeley Software Distribution). Como otro de los productos open source presenta características específicas tales como: alta concurrencia, amplia variedad de tipos nativos, elimina la necesidad de bloqueos explícitos.

Presenta otras características comunes como: llaves foráneas, disparadores ó triggers, herencia de tablas, tipos de datos, operaciones geométricas e integridad transaccional. "PostgreSQL es concebido como una arquitectura cliente / servidor, lo cual requiere procesos separados para cada cliente y servidor, y diversos procesos de ayudante. Además los usuarios pueden crear sus propios tipos de datos, que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL.". (Novo, Marrero, 2008)

2.8.3 Oracle

Sistema de gestión de base de datos multiplataforma creado por Oracle Corporation bajo licencia privativa. Está considerado como uno de los SGBD más completos debido a su: (1) Soporte de transacciones, (2) Estabilidad, (3) Escalabilidad y (4) Soporte multiplataforma. Aunque es considerado el más completo de los sistemas gestores de bases de datos del mundo y está muy ligado a grandes sistemas de búsqueda internacionales, actualmente Oracle enfrenta la competencia de otros SGBD como PostgreSQL, MySQL o Firebird.

2.9 Selección del SGBD a utilizar

Tras lo anteriormente planteado se procede a seleccionar como SGBD que soporte el desarrollo de la aplicación Desktop propuesta, a PostgreSQL.

Se afirma esto debido a que PostgreSQL es un SGBD libre y multiplataforma que centra su estrategia en un sistema llamado MVCC o Acceso Concurrente Multiversión (en inglés, Multi Version Concurrent Access) el cual "permite que mientras un proceso esté escribiendo en una tabla, otros accedan a la misma tabla, siempre se obtiene una versión consistente de lo último a lo que se le hizo commit". (Novo, Marrero, 2008) Además los usuarios pueden crear sus propios tipos de datos que pueden ser completamente indexables gracias a la infraestructura GiST de PostgreSQL.

Cuenta además, con características que permiten conectarlo fácilmente al IDE NetBeans 6.0 (herramienta donde se implementará la propuesta de solución), viabiliza las prácticas de manejo y control de datos, es asequible para el aprendizaje y uso de los desarrolladores y está considerado como uno de los SGBD más actual y versátil del campus.

2.10 Conclusiones parciales

Como antesala a la construcción de la propuesta de solución se realizó un estudio y selección de las herramientas y tecnologías que permitirán el cumplimiento de los objetivos de los lineamientos arquitectónicos para el soporte de la informatización en la UCI. De ahí que la propuesta de solución se resume de la siguiente forma:

Primeramente mencionar que se decide el empleo de RUP Ultra Light como MDS; se considera muy factible por sus particularidades antes mencionadas. Para el modelado de los artefactos generados durante la modelación y construcción de la herramienta se debe hacer uso del lenguaje UML y se utilizará la herramienta CASE Visual Paradigm v3.4 Enterprise Edition que cumple con los estándares de modelado establecidos y es libre. El IDE NetBeans 6.0 será la plataforma de desarrollo para implementar la propuesta de solución por su flexibilidad y potencia. Es además, una herramienta libre y multiplataforma que utiliza en su entorno al lenguaje de programación Java, el cual es ultraseguro y va acorde con la política de producción establecida en la Facultad 9 de la UCI para el desarrollo de aplicaciones Desktop. El SGBD seleccionado es PostgreSQL en su versión 8.2 ya que posee un alto rendimiento y procesamiento de datos, además de presentar grandes facilidades de integración con el NetBeans y Java como lenguaje de programación. Para el tratamiento de imágenes como apoyo al diseño de la herramienta y el documento se hará uso de otras herramientas tales como: Adobe Photoshop CS3, Microsoft Office Picture Manager y otras.

CAPÍTULO 3
LA PROPUESTA DE SOLUCIÓN

Introducción

El presente capítulo contempla las características que presentará el sistema informático de la propuesta que dará solución a la situación problemática planteada anteriormente.

Haciendo uso de la Metodología de Desarrollo de Software (MDS) seleccionada, (RUP Ultra Light), se comienza el proceso de modelación de la aplicación Desktop desde la captura de requisitos a partir de la definición de sus requerimientos funcionales y no funcionales, se seleccionan los actores que interactuarán con el sistema y se confecciona el modelo de diagramas de casos de uso del sistema (CUS) identificándose en él las relaciones existentes. En vías de un mayor entendimiento se exponen las descripciones textuales de cada CUS.

3.1 Descripción del Sistema Propuesto

Durante la etapa de Inicio de la investigación se realizaron varias encuestas y entrevistas con muestras enmarcadas en los Jefes de polos y Líderes de los proyectos productivos de la facultad 9 respectivamente. De cuatro polos productivos con que cuenta la facultad se encuestaron a sus 4 jefes arrojándose los siguientes resultados:

El 50% de los jefes de polos:

- Cuenta con conocimientos parciales sobre los procesos de medición y control de la calidad a partir del uso de métricas de calidad.

- Presenta conocimiento parcial sobre el concepto de métrica.

El 65% de los proyectos productivos:

- No se rige por el modelo de calidad establecido por la Dirección de Calidad en la UCI.
- Presenta una estructura organizacional inadecuada dada preferentemente por la asignación a una misma persona de más de un rol ya sea este complejo o no.
- No presenta equipos de SQA a cargo de las buenas prácticas de los procesos de medición y evaluación de la calidad. Está realidad discrepa de los parámetros planteados por la MDS RUP y el área de proceso de Medición y Análisis del modelo de calidad CMMI.

En síntesis se puede afirmar que los procesos de aplicación de las encuestas y entrevistas no arrojaron los resultados informacionales esperados constatando la inexistencia de procesos visibles en el entorno de negocio de los clientes. De ahí que el comienzo del Modelamiento del Negocio, en caso de haberse realizado, estaría determinado por la descripción de un Modelo de dominio debido a la ausencia de dichos procesos tan indispensables para la modelación. Como respuesta a tal situación y teniendo en cuenta las particularidades mencionadas con anterioridad en el Capítulo 2 en la selección de RUP Ultra Light como MDS, se obvia por completo la existencia de la etapa de Modelamiento del Negocio y comienza la modelación de la herramienta a partir de la Captura de Requisitos.

3.1.1 Requerimientos del sistema a construir

La captura de requisitos es una de las actividades fundamentales en la fase de Inicio de la modelación de un software. Generalmente el proceso de identificación de los requerimientos parte de los artefactos generados en el Modelamiento del Negocio, aspecto que diverge del sistema actual según lo planteado en el epígrafe anterior. Los requerimientos deben ser validados y verificados para evitar futuros errores en el

desarrollo de la herramienta que se modela. Los mismos conforman las raíces en el surgimiento de los casos de uso del sistema (CUS).

A partir del estudio de los procesos de medición de las diferentes MDS, del área de procesos de Medición y Análisis del modelo de calidad CMMI y haciendo uso de la técnica de lluvia de ideas con los Jefes de los polos productivos de la facultad 9 se identificaron los siguientes requerimientos funcionales y no funcionales:

3.1.1.1 Requerimientos Funcionales (RF)

Los RF son en efecto condiciones ó capacidades que el sistema debe cumplir, los cuales se mantienen de forma invariable sin importar con qué propiedades se relacionen.

El sistema debe permitir:

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 datos de Métricas

R4.1 Insertar datos de Métricas

R4.1.1 Calcular datos de Métricas

R4.2 Eliminar datos de Métricas

R4.3 Modificar datos de Métricas

R5 Generar Reporte

R5.1 Obtener Reporte

R5.2 Exportar Reporte a PDF

R5.3 Imprimir Reporte

R5.4 Guardar Reporte

R5.5 Enviar Reporte por correo electrónico

R6 Obtener historial de proyecto por métricas

3.1.1.2 Requerimientos No Funcionales (RNF)

Para lograr una buena aceptación del cliente para con el producto se identificaron los requerimientos no funcionales, los cuales no son más que propiedades o cualidades que el producto debe tener. Estas cualidades comprenden las características que hacen del sistema un entorno virtual atractivo, seguro, usable, confiable, rápido, etc. Estos atributos son fundamentales para llevar a feliz término el desarrollo de la herramienta.

Para este caso en particular se mencionan a continuación los siguientes RNF: Apariencia, Usabilidad, Soporte, Confiabilidad, Seguridad, Portabilidad, Software y Hardware.

RNF 1 Apariencia:

1.1 El diseño de la aplicación Desktop debe ser sencillo para evitar dificultades en el momento de su uso. El mismo 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 presentar colores contrastantes y ser atractiva al usuario. Estará identificada por un color tenue (baige) que brinde 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 Usabilidad:

2.1 El sistema podrá ser usado por un usuario con conocimientos básicos en tecnología Desktop y el manejo de la computadora.

2.2 El sistema podrá ser usado por un usuario con conocimientos parciales o totales de los procesos de cálculo y recopilación de datos de las métricas de calidad.

RNF 3 Soporte:

3.1 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 4 Confiabilidad:

El sistema debe realizar:

4.1 Un tratamiento adecuado de excepciones y validaciones en las entradas de usuarios.

4.2 Recuperación eficaz ante fallos y errores en un intervalo de tiempo correspondiente a 30 segundos (máximo).

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

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 Portabilidad:

6.1 El sistema debe ser independiente de la plataforma.

RNF 7 Software:

Las estaciones de trabajo clientes deben contener:

- IDE: NetBeans 6.0 instalado.
- Adobe Reader 8 instalado. (Para la lectura de documentos PDF)

La estación de trabajo servidora debe contener:

- Sistemas Operativos: Windows XP/ Windows Vista / Linux. (Multiplataforma)
- Gestor de Base de Datos: PostgreSQL 8.2 instalado.

RNF 8 Hardware:

Las estaciones de trabajo clientes deben contener:

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

La estación servidora de Datos debe contener:

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

3.1.2 Descripción de los actores del sistema

Después de identificarse los requerimientos funcionales y no funcionales del sistema se procede a la selección de los actores del sistema. Un actor no pertenece al sistema como tal pero puede intercambiar información con el mismo.

Usualmente en cada proceso de evaluación y control de la calidad de los proyectos productivos de nuestra facultad el Grupo de Calidad desempeña ciertas estrategias de trabajo a través de Auditorías, Revisiones formales, Pruebas, etc. Durante estas actividades los miembros del equipo SQA responden al cumplimiento de funciones correspondientes al rol de Responsable SQA, como se le conoce internacionalmente, o de Administrador de la aplicación o producto a evaluar.

Basado en lo anteriormente expuesto, se muestran en la tabla los actores que interactúan con el sistema acompañado de las respectivas justificaciones del rol que juegan durante su interacción con la herramienta de la propuesta de solución.

Nota: Tenga en cuenta que el sistema que se modela está pensado para ser utilizado por responsables SQA de los diferentes proyectos productivos de la Facultad 9.

Actor	Justificación
Administrador	Miembro del equipo SQA que además 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.
Responsable SQA	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. Nota: El líder de proyecto puede comportarse como un responsable SQA en un momento determinado.

Tabla 2: Descripción de los actores del sistema

3.1.3 Casos de Uso del Sistema

Los CU se comportan como los conductores de la arquitectura del sistema. Los mismos, representan acciones que el sistema debe ejecutar pero desde la óptica del cliente. El sistema cuenta con 17 requerimientos funcionales que se resumen en 6 CU del sistema que contienen secciones.

Seguidamente se representa el Modelo de Casos de Uso del Sistema (MCUS), el cual "describe un sistema en términos de sus distintas formas de utilización, cada una de las cuales se conoce como un caso de uso". **(Weitzenfeld, 2005)**. Es un modelo que contiene actores, casos de uso y sus relaciones.

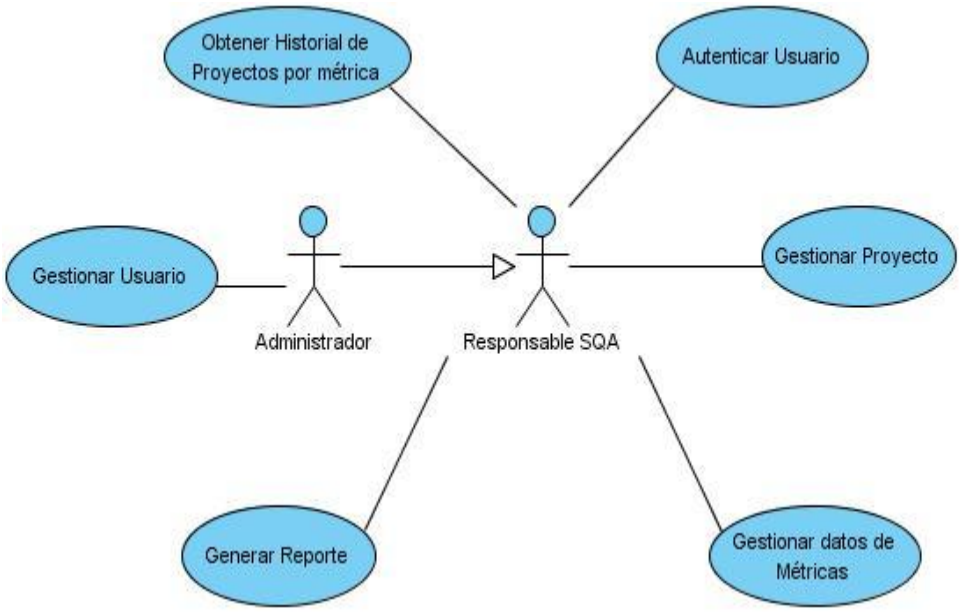


Figura 4: Modelo de CU del sistema

3.1.4 Descripción de los Casos de Uso del sistema

3.1.4.1 Descripción del CU_Autenticar Usuario

Caso de Uso:	Autenticar Usuario
Actores:	Responsable SQA
Resumen:	El CU inicia cuando un responsable SQA desea acceder al sistema y el mismo le pide que se autentique para poder acceder a las funcionalidades y operaciones correspondientes.
Precondiciones:	
Referencias	RF1

CU Asociados	
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El responsable SQA selecciona la opción de “Registrar Usuario”. 3. El responsable SQA ingresa los datos pedidos por el sistema y presiona el botón “Aceptar”.	2. El sistema muestra el formulario con los campos necesarios para la autenticación: *Usuario* y *Contraseña,* además de la opción: • Cambiar contraseña. 4. El sistema verifica que el usuario no esté registrado anteriormente y muestra el mensaje “Usted se ha registrado satisfactoriamente.”
Prototipo de Interfaz	

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3. Presiona el botón “Cancelar”.	4. Muestra el mensaje “Usted no está registrado en el sistema, contacte al administrador.”

Prototipo de Interfaz

Sección “Cambiar contraseña”

Acción del Actor	Respuesta del Sistema
2. El responsable SQA ingresa los datos requeridos para cambiar su contraseña y presiona el botón “Cambiar contraseña”.	1. El sistema muestra el formulario con los campos: *Usuario*, *Contraseña anterior, Nueva contraseña, Confirmar nueva contraseña.* 3. El sistema verifica que los datos del usuario sean correctos y muestra el mensaje “Su contraseña ha sido cambiada”.
Prototipo de Interfaz 	
Flujos Alternos	

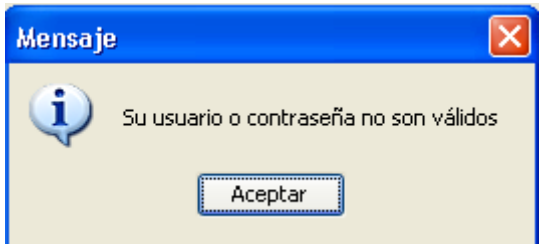
Acción del Actor	Respuesta del Sistema
2. El responsable SQA introduce los datos incorrectamente. 2.1 El responsable SQA oprime el botón "Cancelar".	3. Muestra el mensaje "Su usuario o contraseña no son válidos". 3.1 Muestra la vista anterior.
Prototipo de Interfaz  	
Postcondiciones	Se logra registrar el usuario al sistema.

Tabla 3: Descripción del CU_Autenticar Usuario

3.1.4.2 Descripción del CU_Gestionar Usuario

Caso de Uso:	Gestionar Usuario
Actores:	Administrador
Resumen:	El CU inicia cuando el administrador desea insertar, asignar permisos, modificar permisos, modificar o eliminar a un usuario.
Precondiciones:	El administrador debe estar autenticado en el sistema.

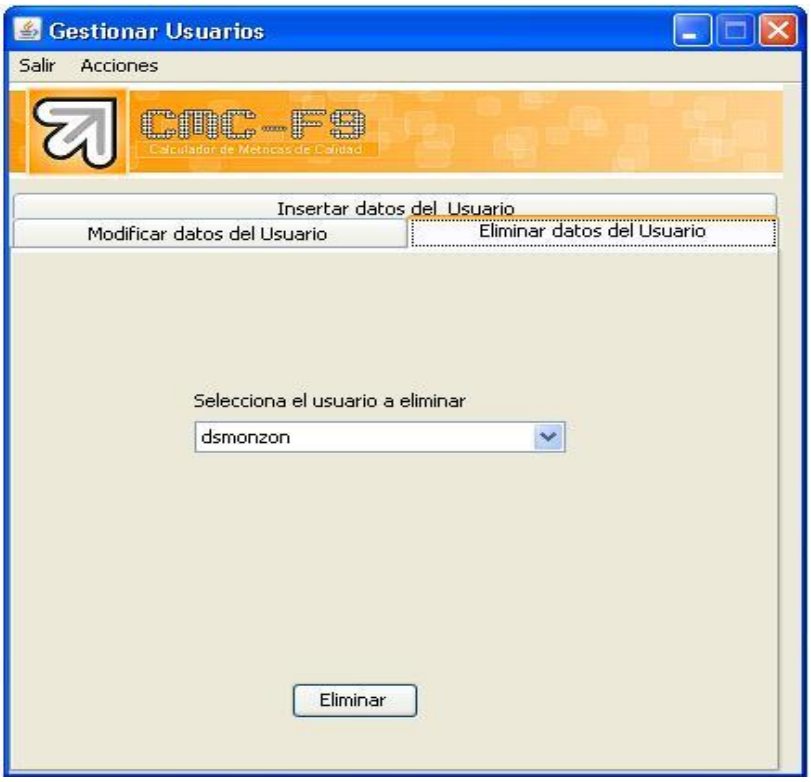
Referencias	RF2, RF2.1, RF2.1.1, RF2.1.2, RF2.2 y RF2.3
CU Asociados	
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El administrador selecciona la opción de “Gestionar Usuario”.	2. El sistema muestra una vista con las opciones: • Insertar Usuario • Eliminar Usuario • Modificar Usuario
3. El administrador selecciona la opción deseada.	4. El sistema ejecuta la opción seleccionada. • Si el administrador desea insertar a un usuario, ir a la Sección “Insertar Usuario”. • Si el administrador desea eliminar a un usuario, ir a la Sección “Eliminar Usuario”. • Si el administrador desea modificar usuario, ir a la

	Sección “Modificar Usuario”.
Sección “Insertar Usuario”	
Acción del Actor	**Respuesta del Sistema**
2. El administrador llena los campos del formulario y presiona el botón “Aceptar”.	1. El sistema muestra el formulario con los campos a llenar correspondiente a esta opción: *Nombre, Usuario, Fecha de Nacimiento, Provincia Brigada, Facultad, No. de Solapin,* además del campo de asignación de Permisos: *Administrador* en caso de que el usuario lo requiera. 3. El sistema ejecuta la opción seleccionada y muestra el mensaje: “Los datos se han insertado satisfactoriamente.”
Prototipo de Interfaz	

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. Presiona la opción “Cancelar”.	3. Muestra la vista anterior.
Prototipo de Interfaz	

Sección “Eliminar Usuario”	
Acción del Actor	**Respuesta del Sistema**
2. El administrador selecciona al usuario al que desea eliminar y presiona la opción “Eliminar”. 4. El administrador presiona el botón “Aceptar”.	1. El sistema muestra un listado con todos los usuarios registrados en el sistema. 3. El sistema muestra el mensaje “¿Está seguro que desea eliminar al usuario seleccionado?” 5. El sistema la acción correspondiente, muestra el mensaje: “El usuario se ha eliminado satisfactoriamente.” y actualiza el listado.
Prototipo de Interfaz	



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
4. El administrador oprime el botón “Cancelar”.	5. Si el administrador no está seguro de eliminar al usuario seleccionado, volver a la vista anterior.
Prototipo de Interfaz	
Sección “Modificar Usuario”	
Acción del Actor	**Respuesta del Sistema**

2. El administrador selecciona al usuario al cual desea modificar. 4. El administrador modifica los datos del usuario y presiona el botón “Modificar”. 6. El administrador oprime el botón “Aceptar”.	1. El sistema muestra un listado con todos los usuarios registrados en el sistema. 3. El sistema muestra el formulario con los campos a modificar correspondientes a esta opción: *Nombre, Usuario, Fecha de Nacimiento, Provincia Brigada, Facultad, No. de Solapin* y el campo de asignación de permisos a modificar: *Administrador.* 5. El sistema muestra el mensaje “¿Está seguro que desea modificar los datos del usuario?” 7. El sistema guarda los cambios y actualiza el listado.
Prototipo de Interfaz	

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
6. El administrador oprime el botón “Cancelar”.	7. Volver a la vista anterior.
Prototipo de Interfaz	
Postcondiciones	Se logró insertar (asignar o modificar permisos de usuarios), modificar y eliminar un usuario.

Tabla 4: Descripción del CU_Gestionar Usuario

3.1.4.3 Descripción del CU_Gestionar Proyecto

Caso de Uso:	Gestionar Proyecto
Actores:	Responsable SQA
Resumen:	El CU inicia cuando un responsable SQA desea insertar, eliminar o modificar los datos de un proyecto productivo.
Precondiciones:	El responsable SQA debe haberse autenticado.
Referencias	RF3, RF3.1, RF3.2 y RF3.3
CU Asociados	
Prioridad	Secundario

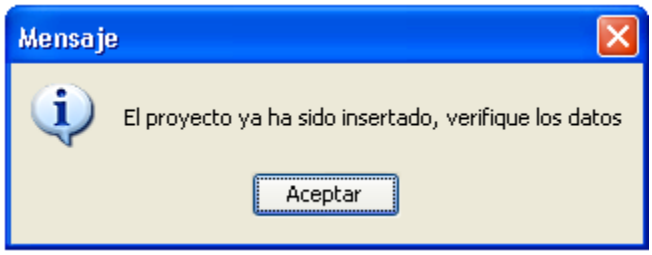
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El responsable SQA selecciona la opción de “Gestionar Proyecto”.	*2.* El sistema muestra una vista con las opciones: • *Insertar Proyecto* • *Eliminar Proyecto* • *Modificar Proyecto*
3. El responsable SQA selecciona la	4. El sistema ejecuta la opción

opción deseada.	seleccionada: • Si el responsable SQA desea insertar los datos de un proyecto, ir a Sección “Insertar Proyecto.” • Si el responsable SQA desea eliminar los datos de un proyecto, ir a Sección “Eliminar Proyecto.” • Si el responsable SQA desea modificar los datos de un proyecto, ir a Sección “Modificar Proyecto.”
Prototipo de Interfaz 	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

3. El responsable SQA presiona el botón “Cancelar”.	4. El sistema vuelve a la vista anterior.
Prototipo de Interfaz	
Sección “Insertar Proyecto”	
Acción del Actor	**Respuesta del Sistema**
2. El responsable SQA ingresa los datos pedidos por el sistema y presiona el botón “Guardar”.	1. El sistema muestra el formulario con los campos necesarios para la opción: *Nombre del proyecto, id. De proyecto, Cantidad de integrantes, Cliente, Duración en meses, Nombre del Líder de Proyecto.* 3. El sistema verifica que el proyecto no esté insertado con anterioridad y muestra el mensaje “Los datos se han insertado satisfactoriamente.”
Prototipo de Interfaz	

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. El responsable SQA presiona el botón “Cancelar”.	3. Muestra el mensaje “El proyecto ya ha sido insertado, verifique los datos”.

Prototipo de Interfaz



Sección “Modificar Proyecto”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un listado con todos los proyectos insertados en el sistema.
2. El responsable SQA selecciona el proyecto al cual va a modificar sus datos.	*3.* El sistema muestra el formulario con los campos necesarios para la opción: *Nombre del proyecto, id. De proyecto, Cantidad de integrantes, Cliente, Duración en meses, Nombre del Líder de Proyecto.*
4. El responsable SQA modifica los datos deseados y presiona el botón “Modificar datos”.	5. El sistema muestra el mensaje ¿Está seguro que desea modificar los datos del proyecto seleccionado?
6. El responsable SQA presiona el botón “Aceptar”.	7. El sistema modifica los datos del proyecto correspondiente y actualiza el listado de proyectos.
Prototipo de Interfaz	

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
6. El responsable SQA presiona el botón “Cancelar”.	7. El sistema muestra la vista anterior.
Prototipo de Interfaz	
Sección “Eliminar Proyecto”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un listado con

2. El responsable SQA selecciona el proyecto al cual va a eliminar y presiona la opción “Eliminar”. 4. El responsable SQA presiona el botón “Aceptar”.	todos los proyectos insertados en el sistema. 3. El sistema muestra el mensaje ¿Está seguro que desea eliminar el proyecto seleccionado? 5. El sistema elimina el proyecto correspondiente y actualiza el listado de proyectos.
Prototipo de Interfaz  	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
4. El responsable SQA presiona el botón “Cancelar”.	5. El sistema muestra la vista anterior.

Prototipo de Interfaz	
Postcondiciones	Se logra insertar, modificar y eliminar los datos de los proyectos productivos de la facultad.

Tabla 5: Descripción del CU_Gestionar Proyecto

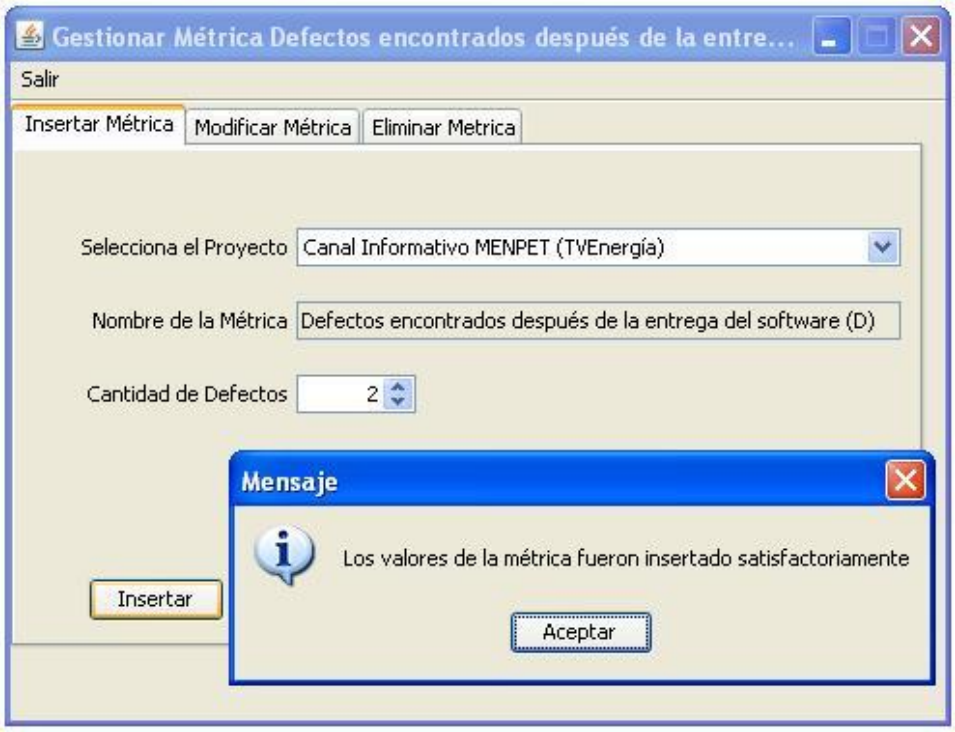
3.1.4.4 Descripción del CU_Gestionar datos de Métricas

Caso de Uso:	Gestionar datos de Métricas
Actores:	Responsable SQA
Resumen:	El CU inicia cuando un responsable SQA desea insertar, eliminar y modificar una métrica.
Precondiciones:	Que el responsable SQA 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 SQA selecciona la opción de “Gestionar datos de la Métrica (Nombre de la métrica)”. 3. El responsable SQA selecciona la opción deseada.	*2.* El sistema muestra una vista con las opciones: • *Insertar datos de Métricas* • *Modificar datos de Métricas* • *Eliminar datos de Métricas* 4. El sistema ejecuta la opción seleccionada: • Si el responsable SQA desea insertar los datos de una métrica, ir a Sección “Insertar datos de métricas.” • Si el responsable SQA desea eliminar los datos de una métrica, ir a Sección “Eliminar datos de métricas.” • Si el responsable SQA 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. El sistema vuelve a la vista anterior.
Prototipo de Interfaz	
Sección “Insertar datos de Métricas”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un formulario con campos que contienen las variables

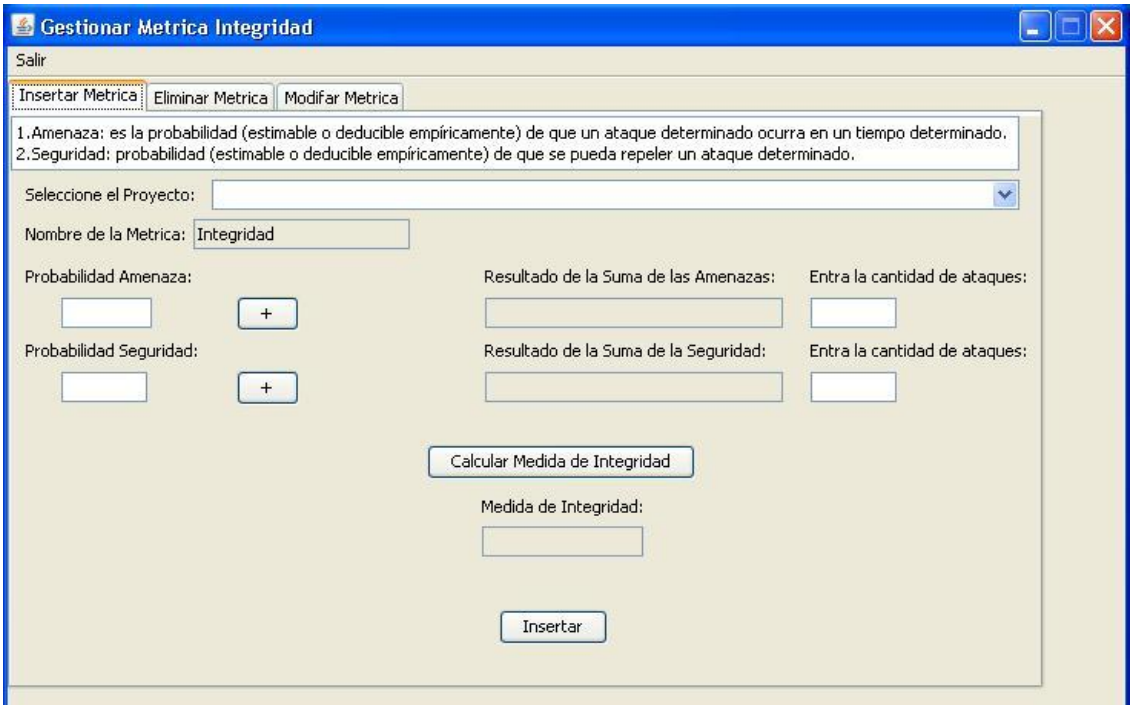
	correspondientes a la métrica seleccionada para insertar los datos.
2. El responsable SQA introduce los datos de la métrica seleccionada correctamente. 4. El responsable SQA oprime el botón “Insertar”.	3. El sistema verifica que los datos estén correctos. • Si la métrica es indirecta el sistema ir a 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.
Prototipo de Interfaz	



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. El responsable SQA presiona el botón “Cancelar”.	3. Muestra la vista anterior.
Prototipo de Interfaz	
Sección “Calcular datos de Métricas”	
Acción del Actor	**Respuesta del Sistema**
1. El responsable SQA oprime el botón “Calcular” para calcular las variables de	2. El sistema ejecuta la acción seleccionada y muestra los

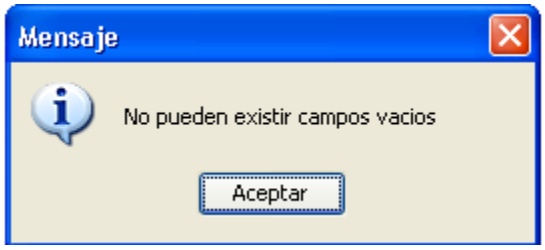
la métrica en cuestión. 3. Volver a inciso 4. de la Sección “Insertar datos de métricas”.	resultados del cálculo de la métrica.

Prototipo de Interfaz



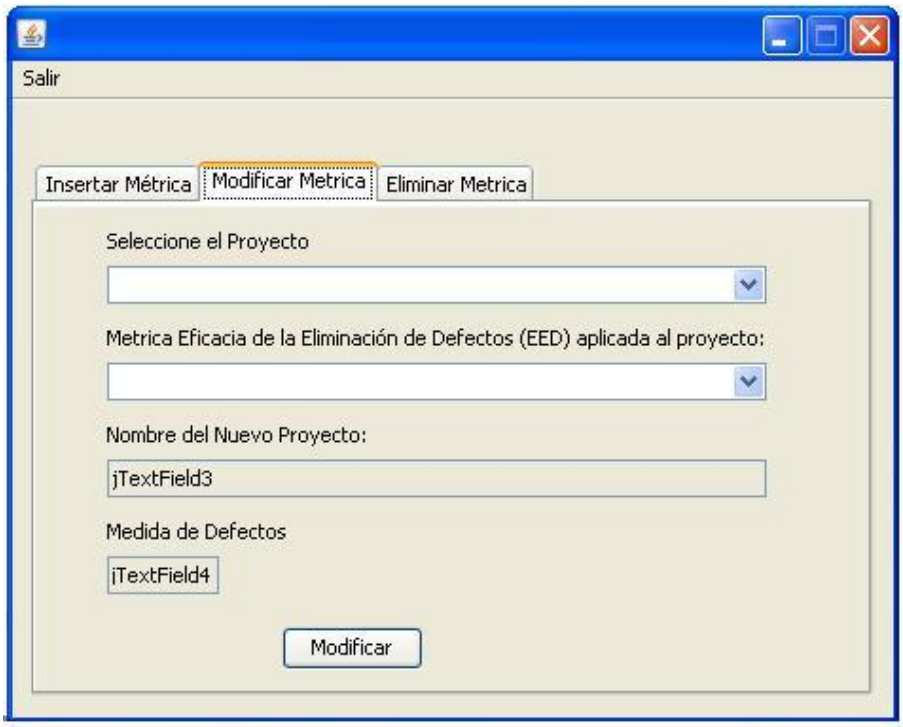
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
1. El responsable SQA no oprime el botón “Calcular” o deja de introducir algún dato.	2. El sistema muestra el mensaje de error: “No pueden existir campos vacíos”.

Prototipo de Interfaz



Sección "Modificar datos de Métricas"	
Acción del Actor	**Respuesta del Sistema**
2. El responsable SQA selecciona la métrica a la cual quiere modificar los datos y presiona el botón "Aceptar".	1. El sistema muestra un listado con todas las métricas de calidad escogidas. *3.* El sistema muestra un formulario con campos que contienen las variables correspondientes a la métrica seleccionada para modificar los datos.
4. El responsable SQA modifica los datos deseados y presiona el botón "Modificar". 6. El responsable SQA presiona el botón "Aceptar".	5. El sistema muestra el mensaje ¿Está seguro que desea modificar los datos de la métrica seleccionados? 7. El sistema ejecuta la opción aceptada y actualiza los datos de la métrica correspondiente.

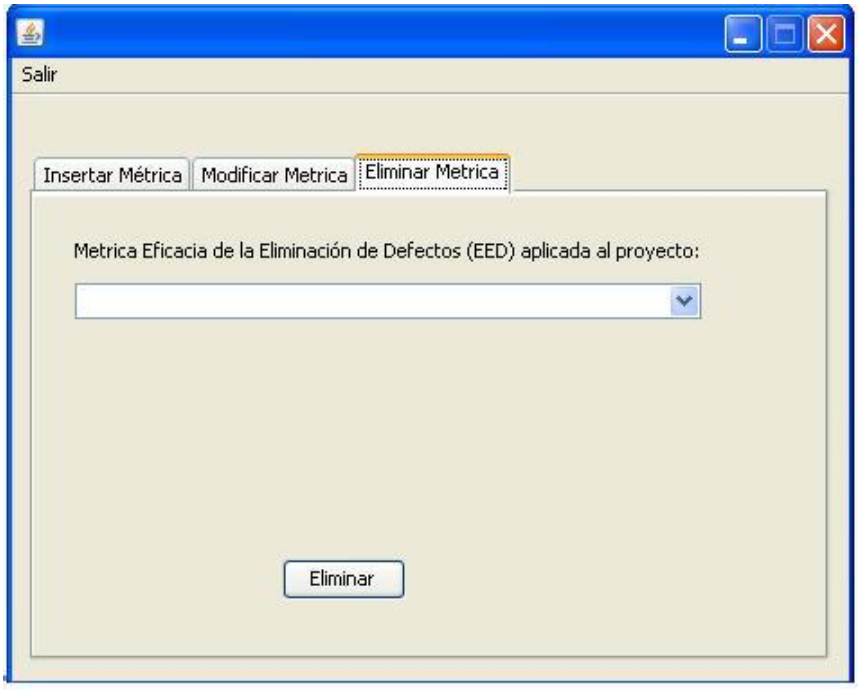
Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
6. El responsable SQA presiona el botón “Cancelar”.	7. El sistema muestra la vista anterior.
Sección “Eliminar datos de métricas”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un listado con todas las métricas de

	calidad escogidas.
2. El responsable SQA 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 SQA presiona el botón "Aceptar".	5. El sistema ejecuta la opción aceptada y actualiza los datos.

Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
4. El responsable SQA presiona el botón	5. El sistema muestra la vista

“Cancelar”.	anterior.
Postcondiciones	Se logra insertar, modificar y eliminar los datos de las métricas de calidad.

Tabla 6: Descripción del CU_Gestionar datos de Métricas

3.1.4.5 Descripción del CU_Obtener Historial de Proyectos por métrica

Caso de Uso:	Obtener Historial de Proyectos por métrica
Actores:	Responsable SQA
Resumen:	El CU inicia cuando un responsable SQA desea obtener un historial de todos los proyectos que han generado una métrica específica.
Precondiciones:	El responsable SQA debe estar autenticado. Los datos de la métrica deben estar insertados en el sistema. Las métricas deben haberse calculado.
Referencias	RF6
CU Asociados	
Prioridad	Crítico
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1. El responsable SQA selecciona la opción de “Obtener Historial”. 3. El responsable SQA seleccionada la métrica correspondiente y oprime el botón “Aceptar”.	*2.* El sistema una vista el listado de las métricas seleccionadas en la propuesta: 4. El sistema muestra un listado con los proyectos que tienen calculada la métrica seleccionada.

Prototipo de Interfaz

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3. Presiona el botón “Cancelar”.	4. Muestra el listado de proyectos vacío.
Prototipo de Interfaz	

	
Postcondiciones	Se logra obtener un historial de los proyectos que tienen calculada las métricas seleccionadas.

Tabla 7: Descripción del CU_Obtener Historial de Proyectos por métrica

3.1.4.6 Descripción del CU_Generar Reporte

Caso de Uso:	Generar Reporte
Actores:	Responsable SQA
Resumen:	El CU inicia cuando un responsable SQA desea obtener, exportar a PDF, imprimir, guardar o enviar por correo electrónico un reporte de los datos de las métricas registradas o calculadas en el sistema.
Precondiciones:	El responsable SQA debe estar autenticado en el sistema.

	Los datos de las métricas que se vayan a reportar deben estar insertados en el sistema. Debe haberse insertado al menos los datos de un proyecto.
Referencias	RF5, RF5.1, RF5.2, RF5.3, RF5.4 y RF5.5
CU Asociados	
Prioridad	Secundario

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El responsable SQA selecciona la opción de “Generar Reporte”.	2. El sistema muestra una vista con las opciones: • *Obtener Reporte.* • *Exportar Reporte a PDF* • *Imprimir Reporte* • *Guardar Reporte* • *Enviar reporte por correo electrónico.*
3. El responsable SQA selecciona la opción deseada.	4. El sistema ejecuta la opción seleccionada: • Si el responsable SQA o

	desea obtener un reporte, ir a "Obtener Reporte". • Si el responsable SQA desea exportar un Reporte a PDF, ir a "Exportar reporte a PDF". • Si el responsable SQA desea imprimir un reporte, ir a "Imprimir Reporte". • Si el responsable SQA desea guardar un reporte, ir a "Guardar Reporte". • Si el responsable SQA desea enviar un reporte vía e-mail, ir a "Enviar Reporte por correo electrónico".
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3. Presiona el botón "Cancelar".	4. Muestra la vista anterior.
Prototipo de Interfaz	
Sección "Obtener Reporte"	

Acción del Actor	Respuesta del Sistema
2. El usuario selecciona la métrica a reportar y oprime el botón “Reportar”.	1. El sistema muestra un listado con las métricas con datos insertados en el sistema. 3. El sistema ejecuta la acción seleccionada por el usuario y genera un documento con los datos de la métrica.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. El usuario oprime el botón “Cancelar”.	3. El sistema muestra la vista anterior.
Sección “Exportar Reporte”	
Acción del Actor	**Respuesta del Sistema**
2. El usuario oprime el botón de “Aceptar”.	1. El sistema muestra las opciones de Exportar documento a formato PFD. 3. El sistema ejecuta la acción seleccionada por el usuario y exporta el documento a

	formato PDF.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. El usuario oprime el botón "Cancelar".	3. El sistema muestra la vista anterior.
Sección "Imprimir Reporte"	
Acción del Actor	**Respuesta del Sistema**
2. El usuario selecciona las opciones de impresión y oprime el botón de "Aceptar".	1. El sistema muestra las opciones de imprimir. 3. El sistema ejecuta la acción seleccionada por el usuario y envía a imprimir al documento.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. El usuario oprime el botón "Cancelar".	3. El sistema muestra la vista anterior.
Sección "Guardar Reporte"	

Acción del Actor	Respuesta del Sistema
2. El usuario selecciona las opciones de de guardar y oprime el botón de “Guardar”.	1. El sistema muestra el cuadro las opciones para guardar el documento de reporte. 3. El sistema ejecuta la acción seleccionada por el usuario.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2. El usuario oprime el botón “Cancelar”.	3. El sistema muestra la vista anterior.
Sección “Enviar Reporte por correo electrónico”	
Acción del Actor	**Respuesta del Sistema**
2. El usuario completa las opciones de envío por correo electrónico y oprime el botón de “Enviar”.	1. El sistema muestra la ventana con las opciones para enviar por correo electrónico. 3. El sistema ejecuta la acción seleccionada por el usuario y envía el documento por correo electrónico.

Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
4. El usuario oprime el botón “Cancelar”.	5. El sistema muestra la vista anterior.
Prototipo de Interfaz	
Postcondiciones	Se logra generar un reporte.

Tabla 8: Descripción del CU_Generar Reporte

Nota: Sólo se presentan los prototipos de interfaz en las descripciones correspondientes a los casos de usos del sistema que se implementaron.

3.2 Conclusiones parciales

La identificación de los RF/RNF, los actores del sistema, los casos de uso del sistema y sus descripciones textuales van a permitir realizar un diseño robusto de la propuesta de solución. Por lo que se resume que: (1) Los RNF permiten trazar las restricciones necesarias que se deben cumplir para el buen funcionamiento de la aplicación que se modela. Son indispensables en la creación del diseño de la arquitectura. (2) Los CUS son las funcionalidades a automatizar. (3) La descripción textual de los mismos permite que se conozca el flujo de eventos de cada una de estas funcionalidades con las cuales interactúan los actores del sistema. Quedaría trazar las metas de la arquitectura, diseñar e implementar la herramienta y realizar las pruebas de la misma.

CAPÍTULO 4
CONSTRUCCIÓN DE LA PROPUESTA DE SOLUCIÓN

Introducción

Sin hacerse esperar llega el capítulo 4, en el cual se abordan aspectos significativos de la arquitectura que sustentará la futura implementación de la herramienta propuesta explicándose brevemente el patrón arquitectónico empleado como complemento a la robustez y estructura organizacional de las características del sistema en específico.

Es aquí donde se crea la propuesta de solución acorde a las necesidades y expectativas del cliente de acuerdo a los requerimientos establecidos para el sistema en desarrollo. 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.

Se representan también 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.

4.1 Descripción de la Arquitectura

El diseño de la arquitectura es una de las etapas fundamentales en el desarrollo del software. A favor de lograr una organización robusta y adecuada de las características del sistema se explican a continuación y de forma breve algunos aspectos de la arquitectura de la herramienta que se modela. Pero antes se deben adquirir acertadas nociones de la importancia de la arquitectura en el proceso de desarrollo del software.

Permitirá definir y detallar los casos de uso críticos para el desarrollo de la herramienta y cuál diseño de arquitectura implementar específicamente.

4.1.1 Patrón arquitectónico empleado

Para poder llevar a cabo un diseño estable y conciso de la arquitectura se necesita de un patrón arquitectónico a seguir. En consecuencia se hace uso del patrón Arquitectura en Capas, el cual se caracteriza por la separación en capas y de forma jerárquica de la Presentación, la Lógica de Negocio y el Acceso a Datos, comportándose como la generalización de la arquitectura Cliente-Servidor. Como estilo de desarrollo en niveles permite en caso de ocurrir cambios, que sólo se realicen las correcciones pertinentes en el nivel requerido sin que se afecte el resto del sistema o el resto de los niveles. Según Jacobson, Booch y Rumbaugh, “este patrón define cómo organizar el modelo de diseño en capas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores… simplifica la comprensión y la organización del desarrollo de sistemas complejos…ayuda a identificar qué puede reutilizarse, y proporciona una estructura que ayuda a tomar decisiones sobre qué partes construir”. **(Jacobson, 1999)**

La figura muestra la estructura en capas del patrón Arquitectura en capas aplicado al sistema por lo que presenta una capa adicional: la Capa de Seguridad. **(Ver detalles en el Anexo 2 del documento)**.

4.1.2 Conformación de la solución propuesta

La arquitectura de la herramienta que se modela centra sus esfuerzos en los casos de uso arquitectónicamente significativos. Uno de sus objetivos fundamentales en el desarrollo del software es definir cuáles son los requerimientos significativos (críticos) para el desarrollo del sistema.

La arquitectura del sistema propuesto estará representada particularmente por las 4+1 vistas arquitectónicas, creadas por Philippe Krutchen en 1995. Están denominadas como vistas de: CU, Lógica, Implementación, Despliegue y Procesos. Debido a la conceptualización del sistema no se hace uso de la vista de Procesos pero sí del resto. Las cuales serán descritas en los epígrafes que siguen a continuación.

4.1.3 Vista de CU

Los CU arquitectónicamente significativos 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. Aseguran además, que la arquitectura del sistema propuesto sea robusta, que se mitiguen los riesgos potenciales y que se cubran las funcionalidades significativas del sistema.

Para priorizar los CU se necesitan saber cuáles son necesarios para las primeras iteraciones en la etapa de desarrollo y cuáles dejar para luego. Los resultados de este proceso se recogen en la presente vista:

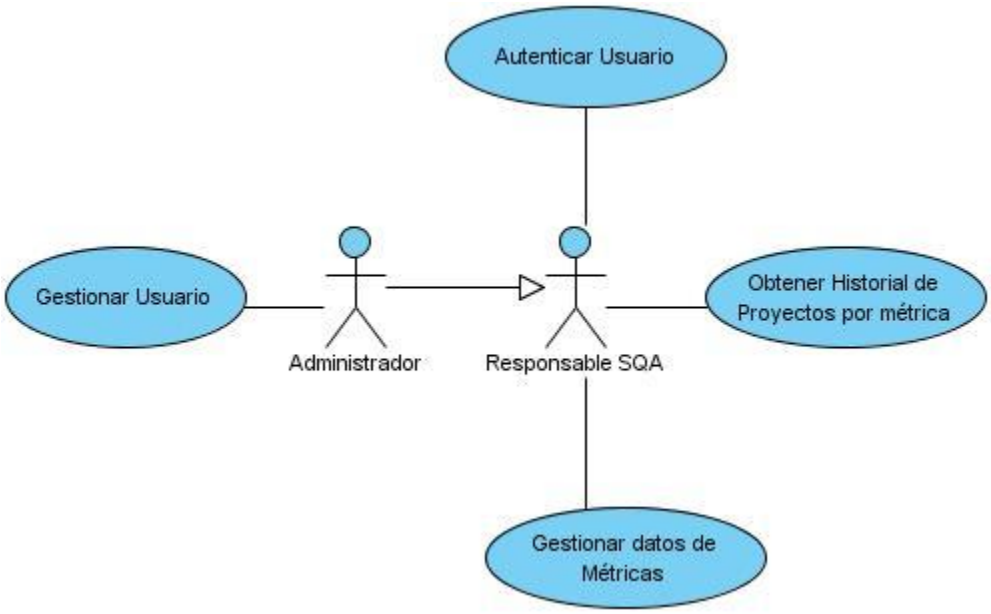


Figura 5: Vista de CU

La vista mostrada anteriormente enmarca su estructura en la compilación de los CU que deben desarrollarse tempranamente en la fase de construcción de la herramienta. Es como tal un fragmento simplificado del MCUS mostrado en la Figura 4.

4.1.4 Vista Lógica

Como parte indispensable de la descripción de la arquitectura, la vista lógica describe las clases más relevantes identificadas durante el proceso de elaboración. Se define la organización de las mismas en subsistemas, los cuales de igual forma estarán localizados en capas.

La siguiente figura muestra la distribución de estos subsistemas dentro de las capas definidas a partir del patrón arquitectónico empleado (Arquitectura en Capas). Consecuente con la selección de los CU arquitectónicamente significativos que se muestran en la Figura 7, es que se realiza la representación del subsistema nombrado Casos de Usos Críticos representado en la Capa de Lógica de Negocio.

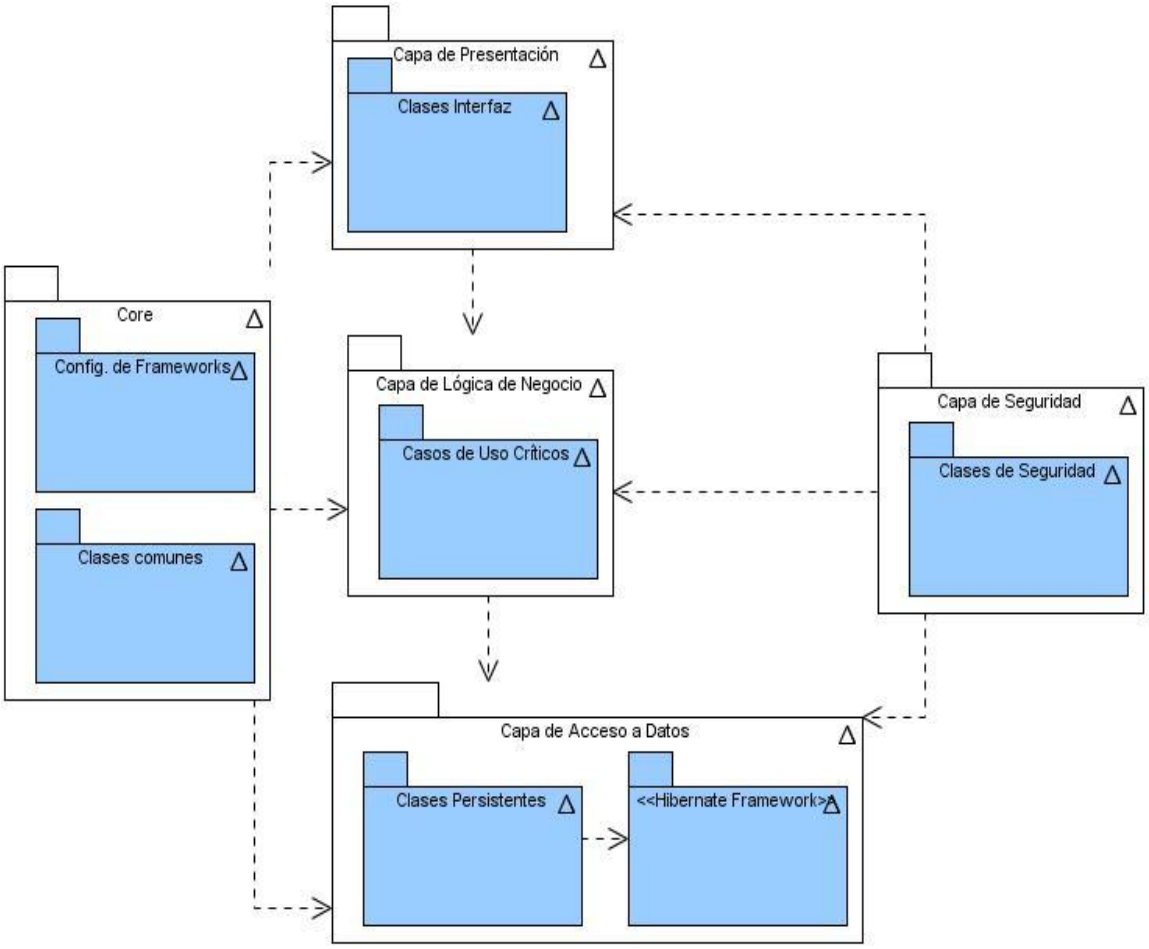


Figura 6: Vista Lógica

Representadas en forma de paquetes emergen las capas presentes en la figura anterior, las cuales cumplen con las siguientes responsabilidades:

Capa de Presentación:

Esta capa reúne todos los aspectos referentes a la interacción de los usuarios con las interfaces del software. Estos aspectos incluyen el manejo de los elementos gráficos de dichas interfaces. Presentación es la capa que muestra el sistema al usuario y le comunica la información necesaria. Se encarga de capturar la información introducida por el usuario dando un mínimo de proceso y sólo se comunica con la capa de negocio.

Su estructura está compuesta por:

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

Capa de Lógica de Negocio:

La capa de Lógica de Negocio contiene "todos los aspectos que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios". (Teruel, 2000). En ella se reciben las peticiones del usuario, residen los programas que se ejecutan en el sistema y se establecen las reglas a cumplir. Para enviar los resultados requiere de la comunicación con la capa superior de Presentación para el recibo de las solicitudes de los usuarios, y para almacenar y recuperar datos de la Base de datos se comunica con la capa inferior Acceso a Datos.

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).

Capa de Acceso a Datos:

Se encarga 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 la capa de negocio y recibe de ella 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, historiales, etc.

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

Capa de Seguridad:

Esta capa centra sus esfuerzos en todas las tareas concernientes a la seguridad del sistema. Aquí se define el alcance y los permisos de cada usuario (rol), desde la autenticación para acceder a la aplicación hasta la administración de su acceso a las diferentes funcionalidades del sistema. Se relaciona directamente con las capas de Presentación, Lógica de Negocio y Acceso a Datos. Es aquí donde se toman las precauciones pertinentes para evitar las inyecciones SQL a la base de datos.

Su estructura está compuesta por:

- Clases de Seguridad: Contiene las clases a cargo de garantizar la seguridad del sistema.

Core:

Contiene todos los componentes comunes para todas las capas. Se ocupa además de la configuración de los frameworks presentes en la construcción de la aplicación. Para este caso en particular sólo presenta la configuración del framework Hibernate.

4.2 Principios de Diseño de la aplicación

Después de observar lo anteriormente expuesto se puede sintetizar que el diseño de la aplicación Desktop se basa en la arquitectura en capas como bien lo reafirma el patrón arquitectónico escogido, que se encarga de separar en niveles la presentación, la lógica y el acceso a datos. A este conjunto de capas se suma la capa de Seguridad de la cual depende el resto para que el usuario pueda interactuar satisfactoriamente con el sistema.

La Presentación es una representación gráfica y atractiva que el usuario ve e interactúa con ella, la Lógica del Negocio reúne los elementos necesarios para recoger las solicitudes enviadas por el usuario y realizar la gestión pertinente en vías de dar una respuesta, cosa tal que ocurre cuando se comunica con la capa de Acceso a datos para almacenar o recuperar la información en cuestión.

El *modus operandi* utilizado por el patrón Arquitectura en capas se presenta de la manera siguiente: (1) El usuario interactúa directamente con la interfaz gráfica, (2) la capa de lógica recibe la notificación de la acción solicitada y la gestiona, (3) se comunica con la capa de acceso a datos solicitando acciones de recuperación o de almacenamiento de la información requerida, (4) recibe respuesta y luego la lógica delega en los objetos de la tarea de desplegar la interfaz de usuario para generar la interfaz apropiada.

4.2.1 Estándares del Diseño de la aplicación

Consecuente con algunos de los RNF expresados al inicio del Capítulo 3 siguen a continuación los estándares por los cuales estará regido el diseño de la herramienta que se construye:

- La aplicación mostrará en su cabecera un banner estático acompañado de un logo. Ambos representados por los colores: naranja degradado (fondo), negro o gris (borde de las figuras) y blanco (letras).
- El fondo de la interfaz gráfica principal y demás formularios es de color baige.
- El menú con las opciones (principales funcionalidades) del sistema se mostrará a la izquierda de la aplicación.
- Los formularios se mostrarán en ventanas pequeñas independientes con los campos (textBox, labels, etc.) alineados en forma de columna.

- Los listados generados en algunas de las operaciones serán mostrados en comboBox en caso de que deba seleccionarse algún objeto presente en ellos.

4.2.2 Patrones de Diseño empleados

Los patrones de diseño comprenden excelentes y simples soluciones a problemas específicos del diseño orientado a objetos de sistemas en desarrollo. Está probado por estudiosos del tema que estas soluciones funcionan y que pueden reutilizarse de sobremanera.

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)

Con el fin de lograr una modularización y estructuración adecuadas del sistema se hace uso del Patrón GoF (gang of four ó pandilla de cuatro): Fachada. Y de los Patrones GRASP de asignación de responsabilidades: Bajo Acoplamiento, Creador y Controlador. Los cuales se exponen seguidamente.

4.2.2.1 Fachada

El patrón Fachada o Facade, como bien se le conoce, es un patrón estructural de diseño que “proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar”. (Gracia, 2005).

El mismo funciona a partir de la creación en el sistema que se desarrolla de una interfaz común para varios componentes ó interfaces. En este caso, el conjunto de componentes lo conforman las funcionalidades de la solución informática que se construye. Este diseño usualmente incurre en un bajo acoplamiento.

4.2.2.2 Bajo acoplamiento

Cuando estamos en presencia de un bajo acoplamiento en los módulos de un sistema se puede asegurar que se ha logrado una buena distribución de las partes que intervienen en el mismo. Este patrón hace referencia a la acción de que un objeto tenga 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.

4.2.2.3 Creador

Como parte indispensable de las clases identificadas en el diseño del sistema el patrón Creador ayuda a identificar quién es responsable de la creación de nuevos objetos o clases del programa a implementar. Brinda soporte al bajo acoplamiento y permite aprovechar oportunidades de reutilización.

4.2.2.4 Controlador

El patrón Controlador muestra su vigencia en el diseño del sistema a partir del uso de una clase Controladora, la cual se comunica con las interfaces de cada funcionalidad y posteriormente con cada clase persistente de la capa de Acceso a Datos. En caso contrario las clases interfaces se comunicarían directamente con las clases entidades presentes en la capa de Acceso a Datos.

4.2.3 Diagrama de Clases del Diseño

El diagrama de clases del diseño no es más que la vista estática del diseño del sistema. El mismo está compuesto por clases interfaces, controladoras y entidades, las cuales se comunican a través de relaciones de dependencia, generalizaciones, etc.

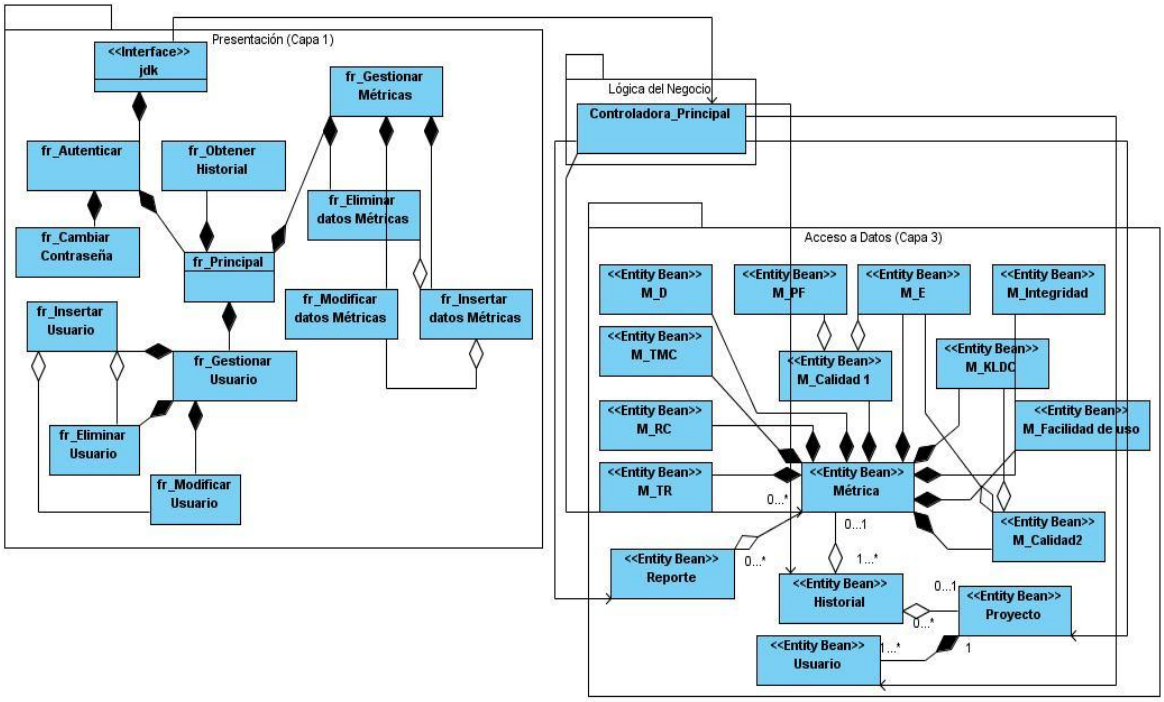


Figura 7: Diagrama de Clases del Diseño

4.3 Diseño de la Base de Datos

La creación del diseño de la base de datos es un proceso al cual no debe subestimarse porque constituye un pilar importante dentro del funcionamiento de la aplicación que se desarrolla. La base de datos soporta todo el manejo de los datos del proceso completo permitiendo la actualización, consulta, inserción o eliminación de datos como respuesta a las diversas peticiones de los usuarios que interactúan con la aplicación.

A continuación se muestra el modelo resultado del diseño de la base de datos creada seguido de la descripción de cada una de las tablas presentes en el mismo:

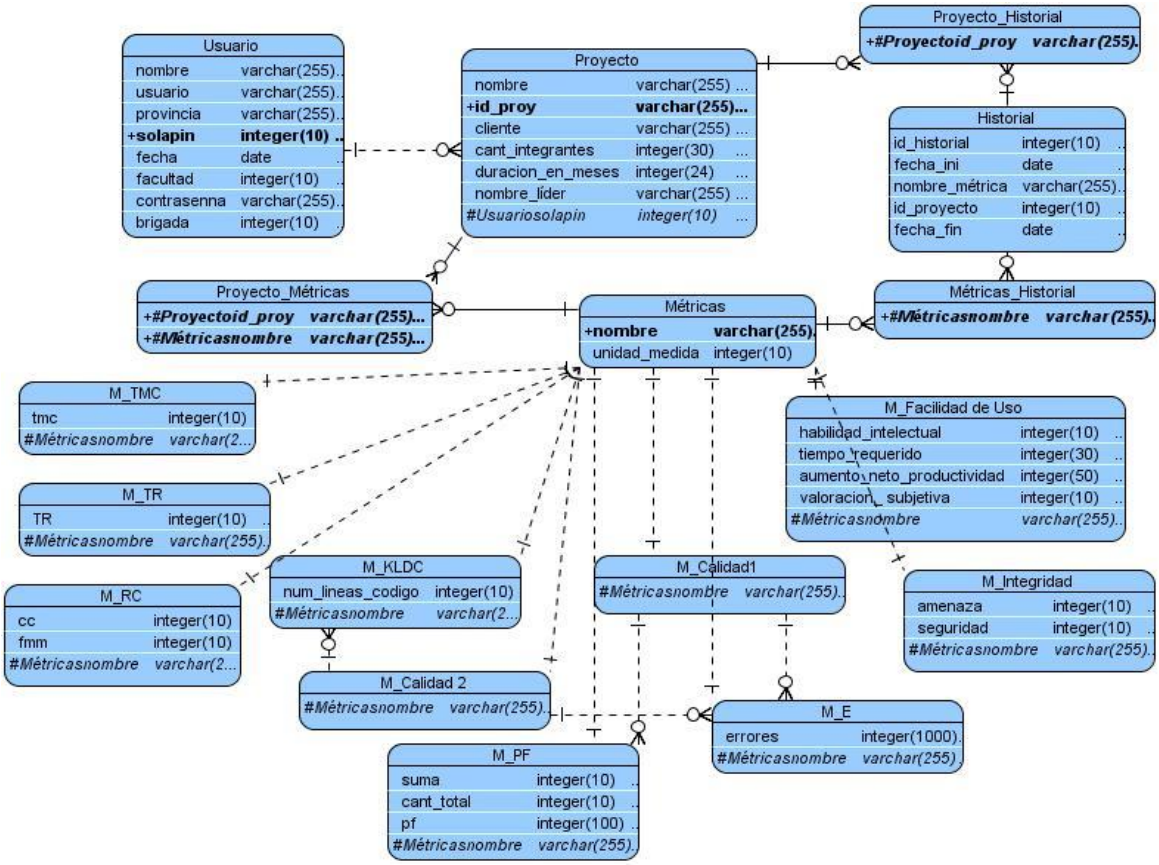


Figura 8: Modelo de Datos

Nombre: Usuario		
Descripción: Guarda los datos de los usuarios del sistema.		
Atributo	**Tipo**	**Descripción**
nombre	Varchar	Guarda el nombre completo del usuario.
usuario	Varchar	Guarda el usuario (rol

		con el que se identifica) del usuario.
provincia	Varchar	Guarda el nombre de la provincia de la facultad del usuario.
solapín	Integer	Conforma la llave primaria de la tabla.
fecha	String	Guarda la fecha en que se realiza la fecha actual en que se realiza la inserción del usuario.
facultad	Integer	Guarda el número de la facultad del usuario.
contrasenna	Varchar	Guarda la contraseña del usuario.
brigada	Integer	Guarda el número de la brigada del usuario.

Tabla 9: Descripción de la tabla Usuario

Nombre: Proyecto		
Descripción: Guarda los datos de los proyectos insertados por los usuarios del sistema.		
Atributo	**Tipo**	**Descripción**
nombre	Varchar	Guarda el nombre completo del proyecto

id_proyecto	Varchar	Guarda el número o código identificador del proyecto.
cliente	Varchar	Guarda el/los nombre(s) completo(s) del/los cliente(s) del proyecto.
cant_integrantes	Integer	Guarda la cantidad de miembros integrantes del proyecto
duracion_en_meses	Integer	Guarda el número en meses de vida (duración) del proyecto
#Usuariosolapin	Integer	Es importado de la tabla Usuario. Está relacionado con su llave primaria.

Tabla 10: Descripción de la tabla Proyecto

Nombre: Métricas		
Descripción: Guarda los datos de las métricas de calidad del sistema.		
Atributo	**Tipo**	**Descripción**
nombre	Varchar	Conforma la llave primaria de la tabla.
unidad_medida	Integer	Guarda la unidad de medida de la métrica de

		calidad.

Tabla 11: Descripción de la tabla Métricas

Nombre: Proyecto_Métricas		
Descripción: Guarda los datos de los proyectos y métricas relacionadas entre sí.		
Atributo	**Tipo**	**Descripción**
#Proyectoid_proyecto	Varchar	Es importado de la tabla Proyecto. Está relacionado con su llave primaria.
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 12: Descripción de la tabla Proyecto_Métricas

Nombre: M_TMC		
Descripción: Guarda los datos de la métrica Tiempo Medio de Cambio (TMC).		
Atributo	**Tipo**	**Descripción**
tmc	Integer	Guarda el valor de la métrica Tiempo medio de cambio.

#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 13: Descripción de la tabla M_TMC

Nombre: M_TR		
Descripción: Guarda los datos de la métrica Tiempo de respuesta (TR).		
Atributo	**Tipo**	**Descripción**
TR	Integer	Guarda el valor de la métrica Tiempo de respuesta.
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 14: Descripción de la tabla M_TR

Nombre: M_RC		
Descripción: Guarda los datos de la métrica Registro de cambios (RC).		
Atributo	**Tipo**	**Descripción**
cc	Integer	Guarda el número de cambios a funciones o

		módulos con comentarios confirmados.
fmm	Integer	Guarda el número de funciones o módulos modificados.
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 15: Descripción de la tabla M_RC

Nombre: M_KLDC		
Descripción: Guarda los datos de la métrica Miles de líneas de código (KLDC).		
Atributo	**Tipo**	**Descripción**
num_lineas_codigo	Integer	Guarda el valor de la métrica Miles de líneas de código (KLDC).
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 16: Descripción de la tabla M_KLDC

Nombre: M_ PF		
Descripción: Guarda los datos de la métrica Puntos de Función (PF).		
Atributo	**Tipo**	**Descripción**
suma	Integer	Guarda el valor de la suma de un conjunto de atributos que se insertan en el sistema.
cant_total	Integer	Guarda el valor de algunos atributos que deben ser insertados en el sistema.
pf	Integer	Guarda el valor de la métrica Puntos de Función (PF).
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 17: Descripción de la tabla M_PF

Nombre: M_E		
Descripción: Guarda los datos de la métrica Errores encontrados antes de la entrega del producto (E).		
Atributo	**Tipo**	**Descripción**

errores	Integer	Guarda el valor de la métrica Errores encontrados antes de la entrega del producto (E).
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 18: Descripción de la tabla M_E

Nombre: M_Calidad1		
Descripción: Guarda los datos de la métrica Calidad1.		
Atributo	**Tipo**	**Descripción**
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 19: Descripción de la tabla M_Calidad1

Nombre: M_Calidad2		
Descripción: Guarda los datos de la métrica Calidad1.		
Atributo	**Tipo**	**Descripción**
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está

		relacionado con su llave primaria.

Tabla 20: Descripción de la tabla M_Calidad2

Nombre: Historial		
Descripción: Guarda los datos de los historiales obtenidos en el sistema.		
Atributo	**Tipo**	**Descripción**
id_historial	Integer	Guarda el número que identifica al historial.
fecha_ini	String	Guarda la fecha en que se obtuvo el historial
fecha_fin	String	
nombre_métrica	Varchar	Guarda el nombre completo de la métrica.
id_proyecto	Integer	Guarda los números que identifican los proyectos que tienen calculadas esas métricas.

Tabla 21: Descripción de la tabla Historial

Nombre: Proyecto_Historial
Descripción: Guarda los datos de los proyectos que contiene un

historial.		
Atributo	**Tipo**	**Descripción**
#Proyectoid_proyecto	Varchar	Es importado de la tabla Proyecto. Está relacionado con su llave primaria.

Tabla 22: Descripción de la tabla Proyecto_Historial

Nombre: Métricas_Historial		
Descripción: Guarda los datos de las métricas que contiene un historial.		
Atributo	**Tipo**	**Descripción**
#Métricasnombre	Varchar	Es importado de la tabla Métricas. Está relacionado con su llave primaria.

Tabla 23: Descripción de la tabla Métricas_Historial

4.4 Generalidades de la Implementación

El presente acápite agolpa su contenido en aspectos generales de la implementación de la propuesta de solución. Aquí se muestran las vistas de Despliegue e Implementación por las cuales se regirá la construcción de la herramienta de software en desarrollo.

Los estándares permiten realizar una solución informática bajo ciertos protocolos que aseguran la calidad e integridad de la misma. En vistas de lograr una implementación

de la herramienta que se modela sólida se identifican los siguientes **estándares de codificación** a seguir:

- Los atributos de las clases del diseño estarán separados por el carácter _ o guión bajo en sustitución al carácter de espacio.
- Se hará uso de objetos (instancias) para lograr una mayor reutilización del código.
- Se declararán clases genéricas con tipado específico del lenguaje de programación que se utiliza: Java.
- Las llaves en los cuerpos de los algoritmos estarán de forma alineada para lograr una mejor organización del código.
- Se implementará un código lo más eficiente posible para evitar redundancias en la sintaxis del código.
- Los nombres que identifiquen a las clases padres identificarán a los de las clases hijas para demostrar la relación estrecha entre ellas.

Como parte de las generalidades de la implementación, se especifica que durante la construcción de la herramienta se implementarán todos los CU arquitectónicamente significativos en una sola clase controladora, ya que muchos dependen de la ejecución satisfactoria de otros.

4.4.1 Vista de Despliegue

El Modelo de despliegue 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. También se utiliza para

visualizar la distribución de los componentes de software en los nodos físicos." **(Piñeiro, 2009)**

El Modelo de Despliegue o vista de Despliegue como bien se le conoce, no es más que el despliegue de los componentes lógicos para los procesadores. Es la forma de ubicar la herramienta de software que se construye en el hardware que la soporta. Según Craig Larman "muestran a los nodos procesadores, la distribución de los procesos y de los componentes". **(Larman, 2006)**

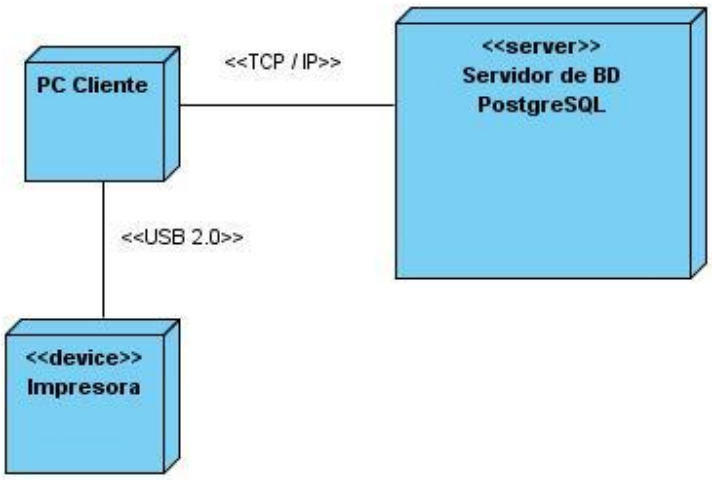


Figura 9: Vista de Despliegue

Seguidamente se realiza una breve explicación de todos los elementos que integran el diagrama anterior, sus características y funciones:

Nombre de elemento	Tipo	Descripción
PC Cliente	Procesador	Nodos clientes del sistema a través de los cuales se accede al mismo mediante el proceso de autenticación. Están distribuidos

		en todos los proyectos productivos de la Facultad 9.
Impresora	Dispositivo	Dispositivo utilizado para satisfacer las solicitudes de impresión de reportes generados por el sistema.
Servidor de Base de datos	Procesador	Contiene la base de datos del sistema. Es aquí donde se encuentra almacenada toda la información que se necesita y se almacenan todos los datos introducidos través de la aplicación.
USB 2.0	Puerto de Conexión	Es utilizado frecuentemente para conectar diferentes clases de dispositivos con un ordenador, en este caso se utilizará para conectar impresoras a los nodos clientes. Se caracteriza por la alta velocidad en la trasferencia de los datos.
TCP/IP	Protocolo	Protocolo utilizado usualmente para conectar los nodos clientes al servidor de base de datos. Se caracteriza por la rápida e íntegra transmisión de la de la información en paquetes.

Tabla 24: Descripción de los elementos del Diagrama de Despliegue

4.4.2 Vista de Implementación

Usualmente un modelo de implementación encierra los componentes del software en construcción que serán implementados y las relaciones existentes entre ellos. Su propósito incide en la captura de las decisiones arquitectónicas hechas por el implementador.

La vista de implementación es útil para:

Calcular el trabajo de implementación de los individuos y equipos.

Calcular la cantidad de código a ser desarrollado, modificado, o eliminado.

Planificar el reúso a gran escala.

Considerar las estrategias de liberación. **(López, 2008)**

En el caso particular de la vista que se muestra a continuación, los componentes aparecen distribuidos en subsistemas respetando el papel que jugará cada uno de acuerdo a la estructura de la arquitectura del sistema.

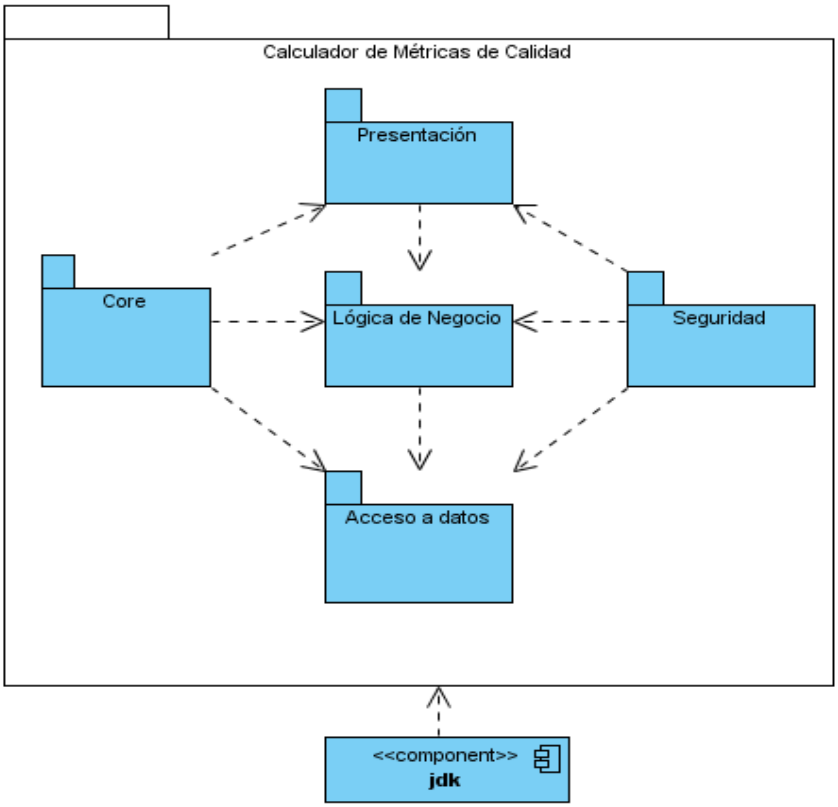


Figura 10: Vista general de Implementación

Debajo se muestran los diferentes componentes presentes en cada uno de los subsistemas representados anteriormente:

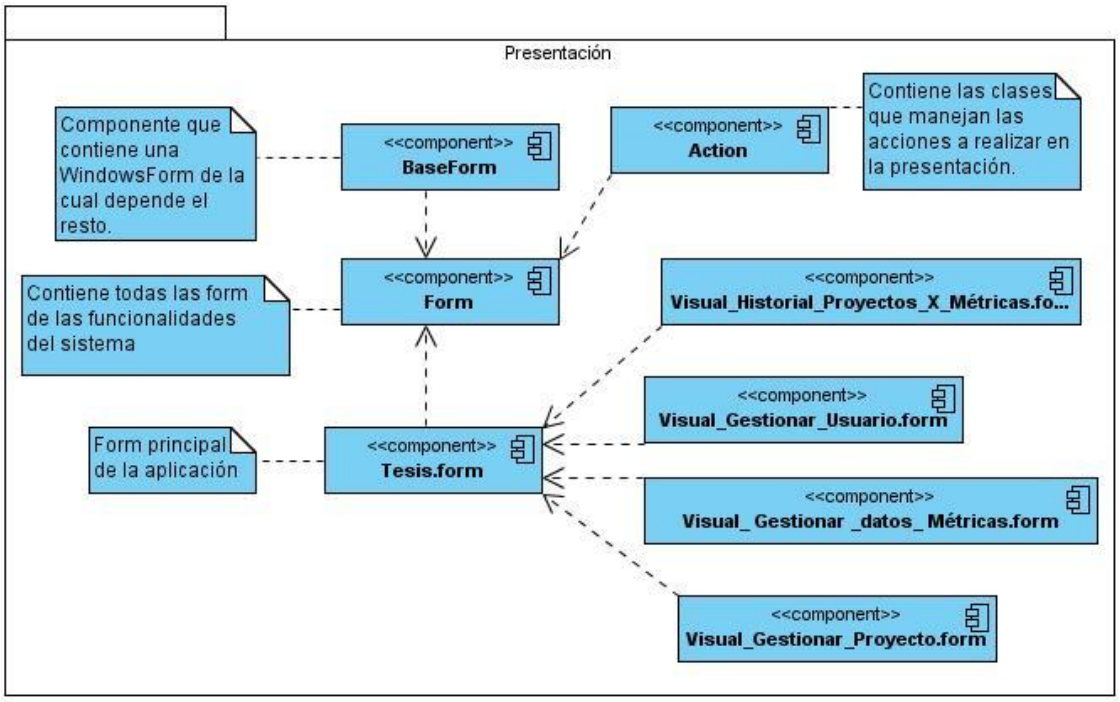


Figura 11: Componentes de la capa de Presentación

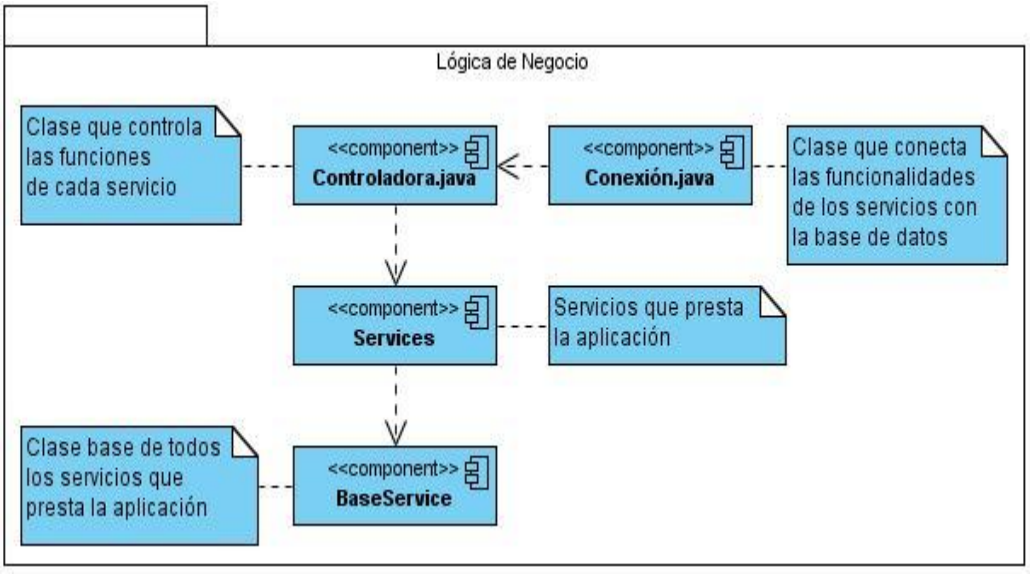


Figura 12: Componentes de la capa de Lógica de Negocio

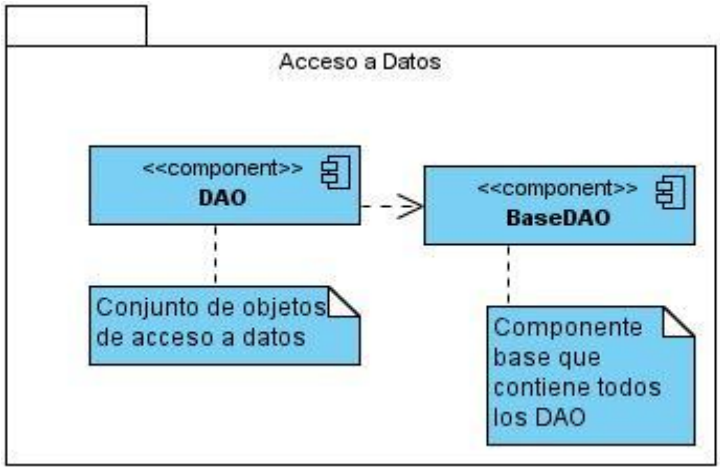


Figura 13: Componentes de la capa de Acceso a Datos

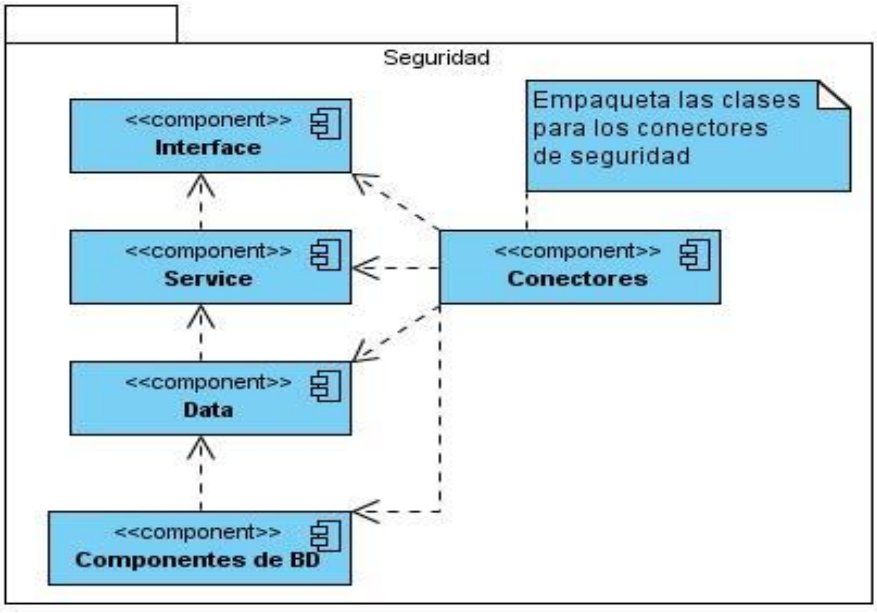


Figura 14: Componentes de la capa de Seguridad

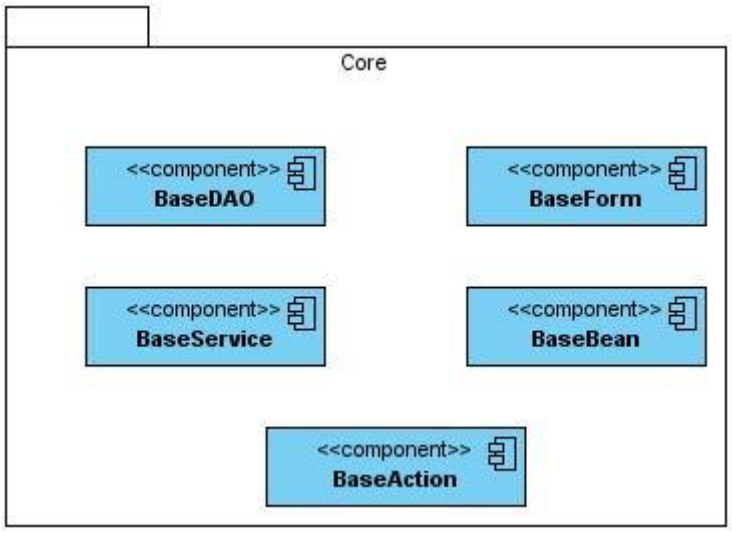


Figura 15: Componentes de la capa Core

4.5 Prueba del sistema propuesto

Según Elsa Ramírez las pruebas de software eran consideradas anteriormente "sólo una actividad que realizaba el desarrollador para encontrar fallas en sus productos; con el paso de los años se ha determinado la importancia que tienen para garantizar el tiempo, el costo y la calidad del producto, de tal forma que actualmente son un proceso cuyo propósito principal es evaluar en todo momento la generación del software respecto de los requerimientos establecidos al inicio". **(Ramírez, 2008).**

Con el propósito de no dejar pasar posibles detalles obviados en las descripciones textuales de CU del sistema o posibles errores de la aplicación ya culminada se accede a realizar Pruebas de Aceptación basadas en la técnica de Caja Negra. Dicha técnica comprende la ejecución de "una prueba del comportamiento observable externamente del sistema". **(Jacobson, 1999)**.

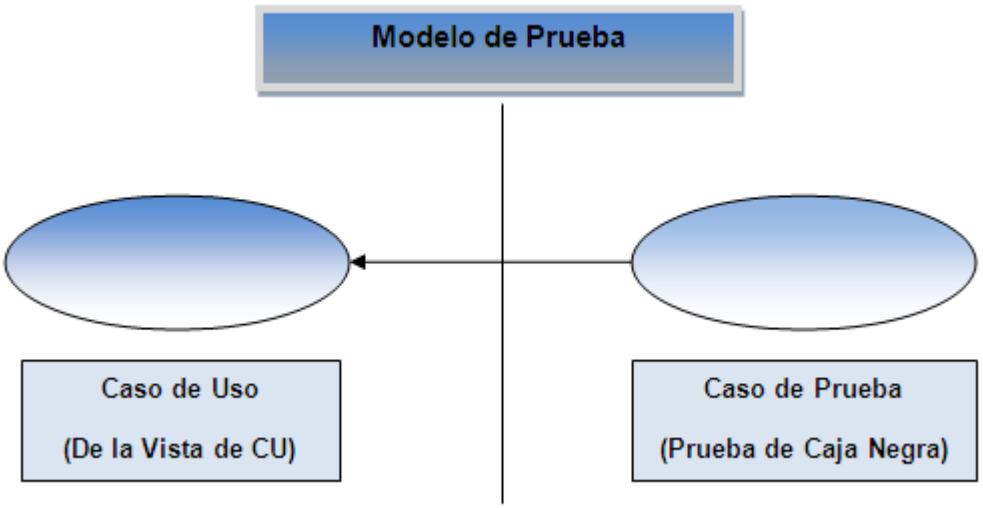


Figura 16: Modelo de Prueba aplicado a Caso de Uso

Con carácter mediato se realizarán pruebas de Unidad y Sistema a la aplicación, las cuales están basadas en Caja Blanca y Caja Negra respectivamente.

Por consiguiente se debe establecer un Plan de Prueba que no es más que la ruta a seguir para desarrollar las pruebas del software de una forma organizada, exigente y eficiente bajo restricciones explícitas de alta calidad. "Un plan de prueba traza la clase de pruebas que se han de llevar a cabo". **(Pressman, 2002)**

El mismo está conformado por: los requerimientos que se desea probar, las estrategias a aplicar, la evaluación y el cronograma de las pruebas seleccionadas, además de los casos de prueba diseñados.

4.5.1 Requisitos a probar

Los requerimientos a probar son:

Autenticar Usuario

Gestionar Usuario

Gestionar datos de Métricas

Gestionar Proyecto

Obtener Historial de Proyectos por métrica

Generar Reporte

Nota: Del requerimiento a probar Generar Reporte sólo se probará la sección Obtener Reporte.

4.5.2 Casos de Prueba diseñados

Jacobson, Booch y Rumbaugh afirman que "un caso de prueba es un conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados, desarrollados para un objetivo concreto, tal como probar un camino concreto a través de un caso de uso, o verificar que se cumple un requisito específico… Los defectos hallados se analizan para localizar el problema. Después estos problemas se priorizan y se corrigen por orden de importancia." **(Jacobson, 1999)**

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.

Prueba de Aceptación para el CU Autenticar Usuario.

Caso de Prueba de Aceptación (CPA)
CU: Autenticar Usuario
Nombre del CPA: CP1_ Autenticar Usuario
Descripción: El CP inicia cuando se desea probar que el Responsable SQA o el

Administrador acceden correctamente al sistema. Para hacerlo cuenta con un usuario y contraseña predeterminados. O el Responsable SQA/Administrador en un momento determinado desea cambiar la contraseña.
Condiciones de Ejecución: Se insertan datos válidos correspondientes al usuario.
Resultado Esperado: El Responsable SQA/Administrador es insertado satisfactoriamente y los datos son guardados en la base de datos del sistema.
Evaluación de la Prueba: Satisfactoria

Tabla 25: CP1_Autenticar Usuario

Caso de Prueba de Aceptación (CPA)
CU: Autenticar Usuario
Nombre del CPA: CP1.1_ Cambiar contraseña de usuario
Descripción: El CP inicia cuando se desea probar que el Responsable SQA/Administrador pueda cambiar su contraseña.
Condiciones de Ejecución: Se introducen los datos correspondientes al cambio de la contraseña.
Resultado Esperado: La contraseña es modificada satisfactoriamente y los datos son actualizados en la base de datos del sistema.
Evaluación de la Prueba: Satisfactoria

Tabla 26: CP1.1_ Cambiar contraseña de usuario

Caso de Prueba de Aceptación (CPA)
CU: Gestionar Usuario
Nombre del CPA: CP2.1_ Insertar Usuario CP2.2_ Modificar Usuario CP2.3_ Eliminar Usuario
Descripción: El CP inicia cuando se desea probar que el Administrador puede gestionar (insertar, modificar, eliminar) los datos de los usuarios.

Condiciones de Ejecución: Se introducen los datos válidos correspondientes para insertar el usuario al sistema. Se introducen los datos correspondientes para modificar los datos anteriores del usuario insertado. Se selecciona al usuario que se desea eliminar del sistema.
Resultado Esperado: El usuario es insertado satisfactoriamente y los datos son guardados en la base de datos del sistema. Los datos del usuario son modificados satisfactoriamente y actualizados en la base de datos del sistema. El usuario es eliminado satisfactoriamente del sistema. Se actualiza la base de datos.
Evaluación de la Prueba: Satisfactoria

Tabla 27: CP2_Gestionar Usuario

Caso de Prueba de Aceptación (CPA)
CU: Gestionar datos de Métricas
Nombre del CPA: CP3.1_ Insertar datos de Métricas CP3.1.1_Calcular datos de métricas CP3.2_ Modificar datos de Métricas CP3.3_ Eliminar datos de Métricas
Descripción: El CP inicia cuando se desea probar que el Responsable SQA puede gestionar (insertar, modificar, eliminar) los datos de las Métricas.
Condiciones de Ejecución: Se introducen y calculan los datos válidos correspondientes a la métrica para insertarla en el sistema. Se introducen los datos correspondientes para modificar los datos anteriores de la métrica insertada. Se selecciona la métrica que se desea eliminar del sistema.
Resultado Esperado:

La métrica es insertada satisfactoriamente y los datos son guardados en la base de datos del sistema. Los datos de la métrica son modificados satisfactoriamente y actualizados en la base de datos del sistema. La métrica es eliminada satisfactoriamente del sistema. Se actualiza la base de datos.
Evaluación de la Prueba: Satisfactoria

Tabla 28: CP3_Gestionar datos de Métricas

Caso de Prueba de Aceptación (CPA)
CU: Gestionar Proyecto
Nombre del CPA: CP4.1_ Insertar Proyecto CP4.2_ Modificar Proyecto CP4.3_ Eliminar Proyecto
Descripción: El CP inicia cuando se desea probar que el Responsable SQA puede gestionar (insertar, modificar, eliminar) los datos de los proyectos.
Condiciones de Ejecución: Se introducen los datos válidos correspondientes al proyecto para insertarlo en el sistema. Se introducen los datos correspondientes para modificar los datos anteriores de la proyecto insertado en el sistema. Se selecciona el proyecto que se desea eliminar del sistema.
Resultado Esperado: El proyecto es insertado satisfactoriamente y los datos son guardados en la base de datos del sistema. Los datos del proyecto son modificados satisfactoriamente y actualizados en la base de datos del sistema. El proyecto es eliminado satisfactoriamente del sistema. Se actualiza la base de datos.
Evaluación de la Prueba: Satisfactoria

Tabla 29: CP4_Gestionar Proyecto

Caso de Prueba de Aceptación (CPA)
CU: Obtener Historial de Proyectos por métrica
Nombre del CPA: CP5_ Obtener Historial
Descripción: El CP inicia cuando se desea probar que el Responsable SQA puede obtener un historial de los proyectos que tienen una métrica determinada calculada.
Condiciones de Ejecución: Se selecciona la métrica calculada en cuestión.
Resultado Esperado: Se obtiene un historial con los proyectos que tienen la métrica seleccionada calculada satisfactoriamente y los datos son guardados en la base de datos del sistema.
Evaluación de la Prueba: Satisfactoria

Tabla 30: CP5_ Obtener Historial

Caso de Prueba de Aceptación (CPA)
CU: Generar Reporte
Nombre del CPA: CP6_ Obtener Reporte
Descripción: El CP inicia cuando se desea probar que el Responsable SQA puede obtener un reporte de los datos insertados de una métrica en el sistema.
Condiciones de Ejecución: Se selecciona la métrica insertada en cuestión.
Resultado Esperado: Se obtiene un reporte con todos los datos insertados de la métrica seleccionada satisfactoriamente y se guarda el reporte en la base de datos del sistema.
Evaluación de la Prueba: Satisfactoria

Tabla 31: CP6_Obtener Reporte

4.6 Conclusiones parciales

Tras lo anteriormente planteado se resume que el cumplimiento de los aspectos de la construcción de la propuesta de solución permitió diseñar la arquitectura de la misma, se determinó el Patrón arquitectónico a seguir acompañado posteriormente de las vistas de arquitectura de Lógica, Implementación y Despliegue del sistema. Se estableció el diseño de la herramienta con el diagrama de clases del diseño correspondiente, además de crearse el modelo de la base de datos que soporta el manejo de datos de las funcionalidades a implementar seguido de las generalidades de la implementación. Finalmente se expuso el proceso de pruebas de Aceptación realizado a la propuesta de solución basada en la técnica de Caja Negra.

Conclusiones Generales

Partiendo de que la definición, modelación y construcción de la herramienta de software resultante del presente Trabajo de Diploma es válida, no por los procesos definidos en los proyectos productivos de la Facultad 9 sino por las necesidades que establecen los estándares de calidad y las Metodologías de Desarrollo de Software, y con el objetivo de dar por cumplido el desempeño del mismo se arriba a las siguientes conclusiones:

- El estudio del arte de la investigación permitió conocer la realidad de la aplicación de los procesos de medición a partir del empleo de métricas de calidad en los proyectos productivos de la Facultad 9 de la Universidad de las Ciencias Informáticas. Además de constatar que dicha realidad discrepa de los procedimientos de medición establecidos en las diferentes MDS y el modelo de calidad establecido en la UCI.

- La definición de los conceptos teóricos acordes al dominio del problema permitieron reconocer un grupo de terminologías tradicionales, actuales e históricas de algunas áreas de la Calidad de Software, las métricas del software y específicamente las métricas de calidad.

- La exposición de 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 de calidad para evaluar los diferentes atributos no medibles que permiten saber cuánto ha avanzado un proyecto de desarrollo de software y en qué debe mejorar.

- El estudio y selección de las diferentes tecnologías y herramientas de desarrollo (MDS- RUP Ultra Light, Lenguajes de programación- Java y modelado- UML, SGBD- PostgreSQL, CASE-Visual Paradigm, 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.

- Basado en lo estipulado en la MDS (RUP Ultra Light) la descripción del sistema de la aplicación propuesta permitió capturar los requerimientos funcionales/no funcionales pertinentes, identificar los actores del sistema, los CU del Sistema y sus respectivas descripciones de textuales como antesala al diseño de la arquitectura de software.

- El establecimiento de la arquitectura de la herramienta a partir del patrón arquitectónico: Arquitectura en Capas permitió dividir en niveles la herramienta durante su etapa de Construcción. De ahí que se realizara una mejor estructuración por capas de las clases representadas para un Diseño inmediato de la aplicación.

- La muestra de las vistas arquitectónicas permitió diagramar como quedaría el funcionamiento del sistema en las etapas de Diseño e Implementación del mismo.

- Por último la aplicación de Pruebas de Aceptación basadas en la técnica de Caja Negra permitió verificar la existencia de errores posiblemente obviados en el desarrollo de los flujos de trabajo anteriores y validar de cierto modo la funcionalidad de la herramienta de software resultante antes de realizar acciones correspondientes al Despliegue de la misma.

Recomendaciones

Para lograr el desarrollo futuro del presente trabajo de diploma se recomienda que:

- Se aplique a los proyectos productivos de la Facultad 9 la herramienta de software resultante.
- Se realicen cursos de capacitación para los desarrolladores a cargo del área de medición en los proyectos productivos donde se abarque todo el contenido teórico del trabajo de diploma presente con el fin de crear una cultura de control y evaluación de la calidad (medir) en la Facultad 9.
- Se establezca en cada proyecto productivo de la Facultad 9 un equipo SQA a cargo de los procedimientos de medición establecidos por las diferentes metodologías de desarrollo de software.
- Se desarrolle una nueva versión de la fase de construcción de la aplicación desarrollada que cuente con la implementación del resto de los CU del sistema u otras que favorezcan a la calidad y aceptación de la misma.
- Se realicen las pruebas de Unidad y Sistema a la aplicación para identificar errores no percibidos en fases anteriores de su ciclo de vida.
- Se modele y desarrolle una versión web de la aplicación construida.
- Se estudien un grupo de patrones de diseño que permitan optimizar la aplicación en vías de que la misma sea lo más modular posible y las clases cada vez más reutilizables.

Bibliografía

Acosta, Zamora A., Betancourt, Ramírez D., *Propuesta de métricas para evaluar el flujo de trabajo Análisis y Diseño.* Cuba: Universidad de las Ciencias Informáticas. 2008.

Agapea Libros Urgentes. *Agapea Libros Urgentes.* [En línea] Disponible en: http://www.agapea.com/libros/MySQL-isbn-8441515581-i.htm. -isbn-8441515581-i.

Álvarez, C. Z., Metodología de la investigación científica. Santiago de Cuba, Cuba: Centro de Estudio de Educación Superior, Universidad de Oriente. 1994.

Hernández, L. y otros., *El paradigma cuantitativo de la investigación científica*. Ciudad de la Habana, Cuba. Editorial Universitaria (ed. univ). 2005.

Álvarez, M. A., *Migrar una base de datos a MySQL. Desarrollo Web*. [2009] Disponible en: http://www.desarrolloweb.com/articulos/1231.php.

Álvarez de Zayas, C., *Metodología de la Investigación Científica*. Santiago de Cuba, Cuba: Universidad de Oriente, 1995

Ambysoft., *The Agile Unified Process. (AUP).* [2005] [Citado: Febrero de 2009] Disponible en: http://www.ambysoft.com/unifiedprocess/agileUP.html#Overview.

Babylon., *Entorno de Escritorio* [Citado: Marzo de 2009] Disponible en: http://www.babylon.com/definition/Entorno_de_escritorio/Spanish.

Bayarre, H; Hersford, R., *Metodología de la Investigación.* Ciudad de La Habana, Cuba. Editorial Ciencias Médicas. 2004.

Stroustrup, Bjarne., *El lenguaje de programación C++.* Madrid. Addison Wesley.1998.

Computer World., *PostgreSQL affiliates*. [2009] Disponible en: http://www.computerworld.com.au/index.php?id=760310963.

Córdova Software Factory., *¿Qué hace un SQA?* [2009] Disponible en: http://cbasqa.wordpress.com/2008/11/22/que-hace-un-sqa/

Cueva, Lovell J. M., *Calidad de Software.* [2008] Disponible en: http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF. 1999.

Domínguez Y., Guillermo y Lozano, Luz., *El concepto de calidad y su aplicación en entidades educativas*. Disponible en: http://cinterfor.org.uy/public/spanish/region/ampro/cinterfor/publ/papel/18/pdf/cap1.pdf.

EUP., *The Enterprise Unified Process (EUP).* [En Línea: 2004] [2009] Disponible en: http://www.enterpriseunifiedprocess.com/#Figure1.

Fowler, Martin., *Diario de Programación.: La Nueva Metodología.* [En Línea: 2003] [2009] Disponible en: http://www.programacionextrema.org/articulos/newMethodology.es.html.

García F. y otros., *Una Ontología de la Medición del Software*. Disponible en: http://www.info-ab.uclm.es/descargas/thecnicalreports/DIAB-04-02-2/UCLMDIAB-04-02-2.pdf.

GIDIS., *Calidad de Software*. Disponible en: http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF.

Hernández, Ballesteros J. F., Minguet, Melián J., *La Calidad del Software y su medida* .Editorial CERASA. 2003

Hernández, Sampieri R., Fernández, Collado C., Baptista Lucio, P., *Metodología de la investigación.* McGraw-Hill. Tercera ed. 2003.

INEM., *Tendencias actuales y futuras de la calidad en la formación*, *Calidad y formación: binomio inseparable*. INEM Publicaciones, Madrid. 2003.

La Calidad del equipo lógico. Disponible en: http://www.csae.map.es/csi/silice/Auditr12.html.

León, Acosta Y. E., Carrasco, Padrón O., *Análisis y Diseño del Portal de la Revista Cubana de Ciencias Informáticas*. Cuba: Universidad de las Ciencias Informáticas. Junio de 2008.

Masip, David., *Desarrollo web.com.* [En línea: 2006.] [Citado el: 10 de Enero de 2008.] Disponible en: http://www.desarrolloweb.com/articulos/840.php.

Métricas. [Citado: Febrero 2009] Disponible en: http://148.202.148.5/cursos/cc321/fundamentos/unidad2/tema2_1.html.

Modelos, Metodologías y Estándares: Estrategias para alcanzar la calidad. Disponible en: http://gidis.ing.unlpam.edu.ar/downloads/pdfs/TEMA%204.

Negrín, Barroso E., Rodríguez, Andino M., *Las TIC y su papel en los procesos de Innovación en el ámbito económico local.* Disponible en: http://www.bibliociencias.cu/gsdl/collect/eventos/index/assoc/HASH2c37.dir/doc.pdf.

Novo, Rijo A. E., Marrero, González L., *Arquitectura del Sistema Informatizado de Cooperación Internacional.* Cuba: Universidad de las Ciencias Informáticas. Junio 2008.

BIBLIOGRAFÍA

Pérez, Torres Carlos., *IEEE.* [En línea] [Citado el: 25 de junio de 2008] Disponible en: http://www.ewh.ieee.org/r9/guadalajara/boletin/marzo02/vistaimpl.htm

Peña, Lemus Y., Hernández, Díaz Y., *SIMETSE – SIstema de METricas para evaluar el Software Educativo*. Ciudad de la Habana. Cuba: Universidad de las Ciencias Informáticas. 2007.

¿Qué es CMMI? [Citado: febrero 2009] Disponible en: http://www.sei.cmu.edu/cmmi/.

Quintana, Yoandri., *Documento de Arquitectura de Software*. Captura y Catalogación de Medias (CCM). Cuba: Universidad de las Ciencias Informáticas, 2008.

Teleformación. *Historia de los Lenguajes de Programación.* [2008] Disponible en: http://teleformacion.uci.cu/mod/resource/view.php?id=9287&subdir=/Tema_2-Tema_2_Sem_5_Clase_No_Presencial_HI_Historia_de_los_Lenguajes_de_Programacion.pdf.

Un enfoque actual sobre la calidad de software Disponible en: http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm.

Valdes, López L., Leal, Ortiz A., *Sistema de Gestión y Control de Métricas.* Cuba: Universidad de las Ciencias Informáticas. 2008.

Referencias Bibliográficas

1. **Álvarez, J. L. M.** *Aplicación de un Sistema Experto para el desarrollo de Sistema .* 2004.
2. **Álvarez, J.C.G.** *Controles y Métricas Técnicas del Software.* 2007.
3. **Arias, A. C. O., Díaz, D. F., Rodríguez, R. T.** *Métrica para establecer el estado de avance de la etapa de implementación de un proyecto de desarrollo de software.* Universidad de las Ciencias Informáticas. Cuba.
4. **Autores, C. D.** Ongei. [En línea] 2004. http://www.ongei.gob.pe/Bancos/Banco_Normas/Archivos/Guia-evaluac.
5. **Briand, L. y B, C.** *An experimental comparison of the maintainability of objectoriented and structured design documents.* 1996.
6. **Cataldi, Z. y Lage, F.** ITBA. [En línea] 2003. http://www.itba.edu.ar/capis/webcapis/RGMITBA/comunicacionesrgm/c-icie.
7. **Crosby, P.** *Quality is free.* New York : McGraw-Hill., 1970.
8. **Cue, R.** *Entrevista personal.* Ciudad de la Habana, 2007.
9. **Dávila, N. L, y Mejía, A. P**. ***Evaluación de la Calidad de Software en Sistemas de Información en Internet.*** **s.l. : CINVESTAV-IPN.**
10. **Davis, C. J.** *Industrial acceptance of software quality assurance standards.* s.l. : IEEE Journal, 1993.
11. **Deming, W. E.** *Calidad, productividad y competitividad. .* Madrid : Díaz de Santos, 1989.
12. **Dondo, Agustín.** [En línea] 03 de Febrero de 2005. http://www.programacion.com/php/articulo/porquephp/. 2005.
13. **Estévez, I.P.** *Métricas para el control proyectos de software.* Instituto Superior Politécnico “José Antonio Echeverría” , Cuba. 2002.
14. **EUP.** [En línea] 2004. http://www.enterpriseunifiedprocess.com/essays/productionPhase.html.
15. **Feingenbaum, A. V.** *Total Quality Control.* Washington : McGraw Hil, 1961.
16. **Fenton, E. N.** *Software Metrics A rigorous approach.* 1991.

17. **Fowler, M., Sccott, K.** *UML Gota a Gota.* 1999.
18. **Gallego, Vázquez, José Antonio.** *Desarrollo Web con PHP y MySQL.* s.l. : Anaya Multimedia, 2003.
19. **Gilb, T.** *Principles of Software Proiect Management.* s.l. : Addison-Wesley, 1988.
20. **Gonzales, H.D.** *Las Métricas de Software y su Uso en la Región, in Departamento de Ingeniería en Sistemas Computacionales.* s.l. : Universidad de las Américas Puebla, Mayo de 2001.
21. Gracia, Joaquín. [En línea] Mayo de 2005. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.
22. **GSInnova.** [En línea] http://www.rational.com.ar/herramientas/roseenterprise.html.
23. **Guerrero, H. C. .** [En línea] http://hanzcocchi.net/proceso-desarrollo-software-orientado-objetos/. 2007
24. *IFP. Function Point Counting Practices Manual, Release 4.0.* s.l. : International Function Point Users Group (IFPUG), 1994.
25. **Ishikawa, K.** *¿Qué es control Total de Calidad? La modalidad japonesa.* Bogotá : Norma, 1986.
26. **Ishikawa, K.** *¿Qué es el Control Total de la Calidad?: Modalidad Japonesa.* 1998.
27. ISO/IEC. *Information technology - Software product evaluation –Quality characteristics and guidelines for their use, JTC 1 Organization. ISO/IEC 9126 JTC 1/SC 7.* 1991. Vol. DOI.
28. **Jacobson, Ivar., Booch, Grady., Rumbaugh, James.** Capítulos 7, 8 páginas 125-163, 187-202. *"El Proceso Unificado de Desarrollo de Software".* s.l. : Addison Wesley, 1999.
29. **James, P.** *Gestión de la Calidad Total.* Madrid : Prentice-Hall, 1997.
30. **Juran, J.** *Quality Control Handbook.* New York : McGraw Hill, 1961.
31. **Kulpa, Margaret K., Kent A. J.** *Interpreting the CCMI. A Process Improvement Approach.* s.l. : Auerbach Publications. Taylor & Francis Group, 2008.

32. **Larman Craig, B. M. V.** *UML y Patrones: una introducción a análisis y diseño orientado a objetos y al proceso unificado.* s.l. : Prentice-Hall, 2006. pág. 590.
33. **López, Mastrapa Henry Yordan.** *Propuesta de Arquitectura del Sistema de Registro de Visitante en la UCI.* Ciudad Habana, Cuba : Universidad de las Ciencias Informáticas, Junio de 2008.
34. **Maqueda, J. y Llaguno, J. I.** *Marketing estratégico para empresas de servicios.* Madrid : Díaz de Santos, 1994.
35. **Martín, M. A, Bertoa, M. F, Valecillo, A., Orsina L.** *Hacia un Enfoque Semántico para la Catalogación de Métricas.* s.l., España : GIDIS. Departamento de Ciencias de la Computación. Universidad de Málaga, 2004.
36. **McCall J. A., Cavano J. P.** *A Framework for the Measurement of Software Quality, ACM Software Quality Assurance Workshop.* 1978.
37. **Melián, J. M. M., Hernández Ballesteros,J. F.** *La calidad del software y su medida.* s.l. : CERASA, 2003.
38. **Mena, M. Gonzalo.** *ISO 9126-3: Métricas Internas de la Calidad del Producto de Software.* s.l. : Facultad de Informática. Universidad Autóctona de Querétaro, 2006.
39. **Mendoza, S., María A.** [En línea] 07 de Junio de 2004. http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.
40. **MGAR.** [En línea] http://mgar.net/soc/isointro.htm.
41. **Mojena, B.G.** *Procedimiento Propuesto para medir la Calidad en la Gestión de Requisitos.* 2007.
42. **Muñoz, C. C.** [En línea] 2006. http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software. 2006.
43. **Nguyen, A., Kleiner, B. H.** *Technical Report: European company examples of excellent quality management. International Journal Vehicle Design .* 1995. págs. 594-599.
44. **Normas ISO 9000.** [En línea] 2007. http://normas-iso-9000.blogspot.com/2007/12/grandes-modelos-de-calidad-total.html.

45. **Oliva, Agüero D., Cruz, Pichardo O. E.** *Métricas para la evaluación del proceso de desarrollo de Software Educativo.* Ciudad habana, Cuba : Universidad de las Ciencias Informáticas, Junio de 2008.
46. **Pressman, R.** *Ingeniería de Software. Un enfoque práctico.* s.l. : MCGraw Hill, 1993.
47. **Pressman, R.** *Ingeniería de Software. Un enfoque práctico.* s.l. : MCGraw Hill, 2002.
48. **Piñeiro, G. Yordanys.** *Modelo de Despliegue v1.0.* Grupo de Desarrollo de Sistemas de Información Geográfica. Plataforma Soberana LiberGiS. Cuba: Universidad de las Ciencias Informáticas, Mayo de 2009.
49. **Q., Ernesto.** [En línea] http://www.eqsoft.net/presentas/modelos_de_calidad_y_software_libre.pdf.
50. **Ramírez, Elsa.** [En línea] 06 de Marzo de 2008. [Citado el: 15 de Abril de 2008.] http://www.sg.com.mx/content/view/684/1/.
51. **Roselló, E. G. y G., Dacosta J.** [En línea] 1994. http://lsm.dei.uc.pt/ribie/docfiles/txt2003729185619paper-144.pdf.
52. **RUP**. *Rational Unified Process.* 2003.
53. **SEI**. *Capability Maturity Model Integration (CMMISM).* s.l. : Carnegie Mellon University, 2002. pág.
54. **Shoonmaker, S. J.** *ISO 9000 for Engineers and Designers.* s.l. : McGraw Hill, 1997.
55. **Sun Microsystems.** [En línea] http://java.sun.com/javase/technologies/desktop/index.jsp.
56. **Tecnomaestros.** [En línea] http://tecnomaestros.awardspace.com/estandares_iso.php.
57. **Teruel, Alejandro.** [En línea] 15 de Septiembre de 2000. [Citado el: 04 de Mayo de 2009.] http://www.ldc.usb.ve/~teruel/ci3715/clases/arqCapas.html.
58. **Vega, Lebrún C., Rivera Prieto L. S., García, Santillán A.** [En línea] http://www.eumed.net/libros/2008a/351/Metricas%20de%20Calidad.htm. 2008.

59. **Vega, Lebrún C., Rivera, Prieto L. S., García, Santillán A.** [En línea] 2008. http://www.eumed.net/libros/2008a/351/351.zip. 2008.
60. **Ves, Esther de., Cerverón, V.** [En línea] 2006. http://informatica.uv.es/iiguia/2000/BD2/1_0_BD2Tema1_06.pdf. 2006.
61. **Weitzenfeld, Alfredo.** *Ingeniería de Software orientada a objetos con UML, Java e Internet.* México : Thomson Paraninfo S.A, 2005.

Anexos

ANEXO 1

Pasos para el Cálculo de métricas de Puntos de Función:

1. **Los puntos de función se calculan llenando la tabla de la siguiente figura**:

Figura 17: Características que forman parte del Factor de Ponderación para el cálculo de métricas de Puntos de Función

Nota: Las características recogidas en la figura anterior forman parte del ámbito de información y se sintetizan de la siguiente manera:

Número de entradas de usuario: se cuenta cada entrada del usuario que proporcione al software diferentes datos orientados a la aplicación. Las entradas deben diferenciarse de las peticiones, las cuales se contabilizan por separado.

Número de salidas del usuario: se encuentra cada salida que proporciona al usuario la información orientada a la aplicación. En este contexto las salidas se refieren a informes, pantallas, mensajes de error. Los elementos de datos individuales dentro de un informe se cuentan por separado.

Número de peticiones al usuario: una petición está definida como una entrada interactiva que desencadena la generación de algún tipo de respuesta en forma de salida interactiva. Se cuenta cada petición por separado.

Número de archivos: se cuenta cada archivo maestro lógico, o sea una agrupación lógica de datos que puede ser una parte en una gran base de datos o un archivo independiente.

Número de interfaces externas: se cuentan todas las interfaces legibles por la máquina (por ejemplo: archivos de datos, en cinta o discos) que son utilizados para transmitir información a otro sistema. **(IFP,1994)**

2. **Para calcular los puntos de función se utiliza la siguiente relación.**

PF = CUENTA_TOTAL * [0.65 + 0.01 * SUM(fi)]

Donde:

CUENTA_TOTAL es la suma de todas las entradas de PF obtenidas de la Tabla 1.

Fi es el resultado del Procedimiento (3) que sigue debajo.

3. **Evaluar cada factor Fi en escala 0 a 5:**

Evaluación del factor (Fi)					
0	**1**	**2**	**3**	**4**	**5**
Sin Influencia	Incidental	Moderado	Medio	Significativo	Esencial

Tabla 32: Evaluación del factor (Fi) en una escala de 0 a 5.

Para lograr la evaluación debe responder las siguientes preguntas con valores de la escala de 0 a 5:

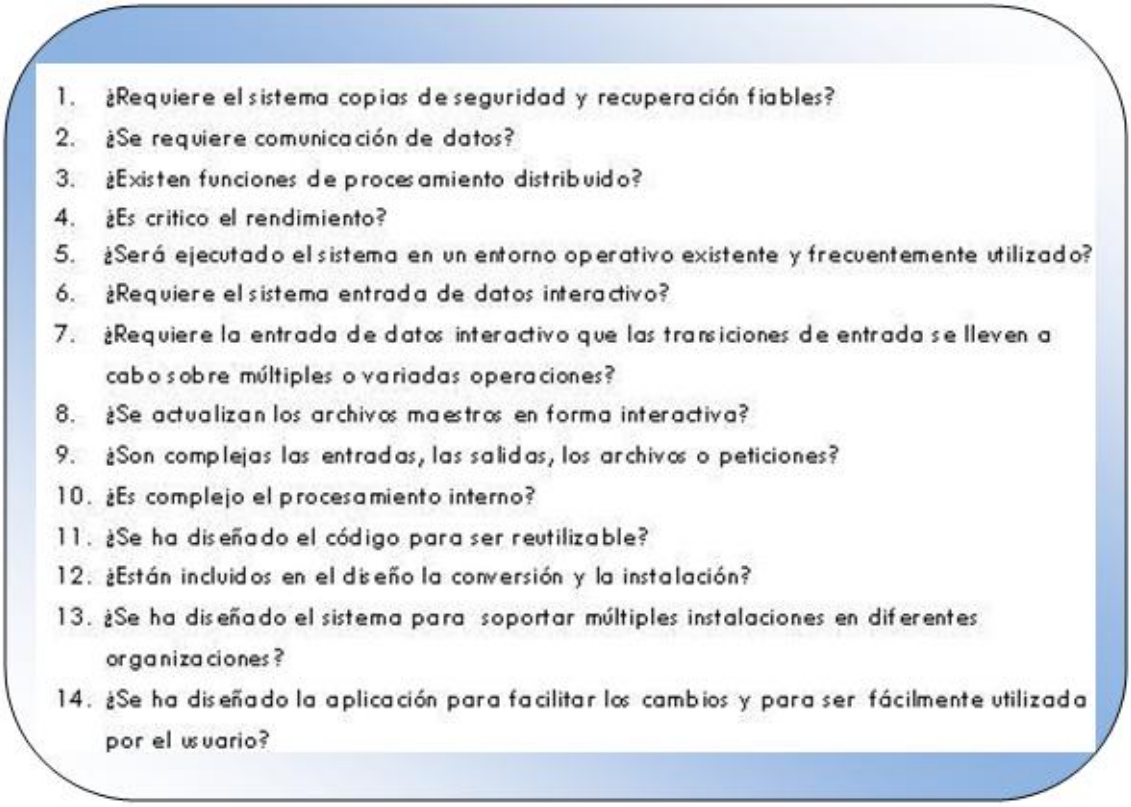


Figura 18: Evaluación de cada una de las preguntas en factores Fi de 0 a 5.

Nota: La suma total de los valores arrojados para cada Fi será el resultado de SUM(Fi) de ahí que permita el cálculo de la ecuación de PF.

ANEXO 2

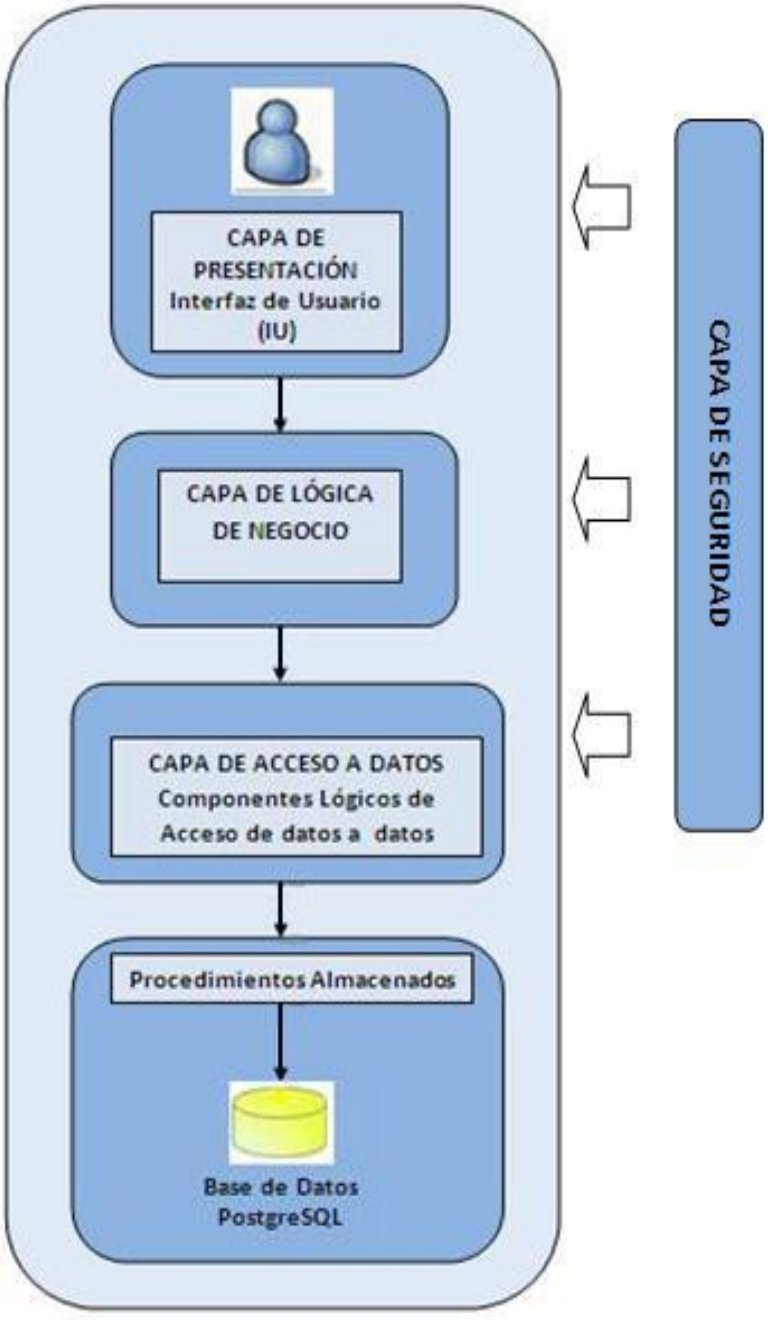


Figura 19: Patrón arquitectónico: Arquitectura en capas

ANEXO 3

Universidad de las Ciencias Informáticas

Encuesta realizada a los Jefes de los polos productivos de la Facultad 9.

Nombre:

Proyecto:

Facultad:

***IMPORTANTE: **Mencionar cada uno de los proyectos que atiende y la metodología aplicada en el proceso de desarrollo de software de cada uno de ellos.**

Proyecto	Metodología de desarrollo de software que utiliza

1. ¿Cómo controlan la calidad del software?

Pruebas_____ **Métricas**_____ **No se controla**_____

Otros_____

En caso que sea otros, mencionarlos_____

2. ¿Qué riesgos tiene una mala calidad?

_____

3. ¿Tienen algún conocimiento de las métricas que se utilizan para la evaluación del software?

Si____ **NO**_____

4. ¿Conoce si se aplican dichas métricas para la evaluación de software en la UCI?

Si____ **No**____

En caso positivo: ¿Cuáles métricas aplican?_____

¿Y en qué etapas del proyecto aplican las métricas?_____

En caso negativo ¿por qué no?_____

5. ¿Es importante la aplicación de las métricas para la evaluación del software?

Sí___ **NO**___

¿Por qué? _____

6. ¿Las métricas son una alternativa para eliminar los errores en los productos?

Sí_____ **No**_____

¿Por qué?_____

7. ¿Ayudan las métricas a mejorar la calidad del software?

Sí_____ **No**_____

8. Entre los niveles organizacionales están:

PSP____ TSP_____ CMM_____

Marque con una cruz el que utilizan

¿Qué importancia otorgan al nivel organizacional utilizado?

Alta____ **Media**____ **Baja**____

¿Por qué?_____

Nota:

(CMM) **Capability Maturity Model**

(PSP) **Personal Software Process**

(TSP) **Team Software Process.**

Elaborado por: Est. Yisel Nadereau Sagarra

Bibliografía consultada: Peña, Lemus Yudisleidys, Hernández, Díaz Yuniersy., *SIMETSE – SIstema de METricas para evaluar el Software Educativo*. [Consultada en: Diciembre 2008] Disponible en: http://bibliodoc.uci.cu/TD/TD_0803_07.pdf. Julio de 2007. Páginas 101-102.

ANEXO 4

Entrevista realizada a los Líderes de los proyectos productivos de la Facultad 9.

Universidad de las Ciencias Informáticas

Facultad 9

Fecha: 19-1-2009

Entrevistados: Líderes de Proyectos Productivos de la facultad

Preguntas a responder:

1. ¿Actualmente se miden los indicadores: coste, esfuerzo, efectividad en el proyecto?
2. ¿Se estimó durante planificación en la fase de Inicio?
3. ¿Existe alguien a cargo del rol de medición?
4. ¿Existe un equipo de SQA en el proyecto?
5. Cantidad de integrantes del equipo de SQA
6. ¿Qué hacen sus integrantes?
7. ¿Qué métricas utilizan?
8. ¿Sabe qué es una métrica? ¿Para qué les sirve?
9. En caso de utilizar métricas, ¿por qué la utilizan?

Glosario de Términos

❖ A

Artefacto: Documentación tangible que es creada, modificada y usada por los desarrolladores al realizar actividades. Un artefacto puede ser un modelo, un elemento de un modelo, un diagrama, o un documento.

Arquitectura: Conjunto de decisiones significativas acerca de la organización de un sistema software. Especifica la estructura, el comportamiento, las restricciones y compromisos de uso, funcionalidad, funcionamiento, flexibilidad al cambio, reutilización, compresión, economía y tecnología, así como por aspectos estéticos del software.

Atributo: Característica del software que pretendemos medir y cuantificar. Ejemplo: esfuerzo, fiabilidad, mantenibilidad, calidad y tamaño, funcionalidad, eficacia, etc.

❖ C

Ciclo de vida del software: Ciclo que cubre 4 fases por las que transita un software.

Cliente: Persona, organización o grupo de personas que se encarga de la construcción de un sistema, ya sea empezando desde cero, o mediante la simplificación de versiones sucesivas.

CMM: Modelo de calidad denominado Modelo de Madurez de la Capacidad del Desarrollo del Software o Capability Maturity Model. Ambiente de trabajo de madurez de procesos de software creado por el Software Engineering Institute (SEI).

CMMI: Modelo de calidad denominado Modelo de capacidad de madurez integrada. Es la versión ampliada y mejorada de CMM.

CP: Caso de Prueba

CU: Caso de Uso

CUS: Caso de Uso del Sistema

❖ D

Diseño (flujo de trabajo): Flujo de trabajo fundamental cuyo propósito principal es el de formular modelos que se centran en los requisitos no funcionales y el dominio de la solución, y que prepara para la implementación y pruebas del sistema.

❖ F

Fase de inicio: Primera fase del ciclo de vida del software.

❖ I

IEEE: Instituto de Ingenieros Eléctricos y Electrónicos o The Institute of Electrical and Electronics Engineer.

Implementación (flujo de trabajo): Flujo de trabajo fundamental cuyo propósito esencial es implementar el sistema en términos de componentes, es decir, código fuente, guiones, ficheros binarios, ejecutables, etc.

ISO: Organización Internacional para la Estandarización o International Standard Organization.

❖ M

MDS: Metodología de Desarrollo de Software.

Métrica: Asignación de un valor a un atributo de una entidad propia del software, ya sea un producto o un proceso.

Métrica del software: es una medida cuantitativa del grado en el que un sistema, un componente o un proceso poseen un determinado atributo.

Métricas de Calidad: es una medida cuantitativa que permite medir los atributos de calidad de un software.

MySQL: Sistema de gestión de base de datos relacional.

❖ P

Producto: Comprende el resultado de una iteración o el ciclo completo del desarrollo del software. Por ejemplo: un código, un diseño, una especificación, etc.

Proyecto: Esfuerzo de desarrollo para llevar un sistema a lo largo de un ciclo de vida.

❖ R

Requisitos funcionales (RF): Se refieren a las funciones específicas que el software debe cumplir.

Requisito no Funcional (RNF): Se refieren a las propiedades del sistema. Especifican restricciones físicas de los RF. Por ejemplo: rendimiento, usabilidad, apariencia externa, portabilidad, mantenibilidad, extensibilidad o fiabilidad.

RUP: Rational Unified Process (Proceso Unificado del Desarrollo de Software). MDS creada por IBM Rational para el desarrollo y construcción de software basado íntegramente en el lenguaje de modelado UML que soporte a la metodología.

RUP Ultra Light: MDS ágil utilizada para el desarrollo del ciclo de vida del software que cuenta con 10 pasos en cuatro fases: Análisis. Diseño, Implementación y Puesta en marcha.

SGBD: Sistema Gestor de Bases de Datos, es el software que permite realizar el manejo y/o la actualización de los datos almacenados en una o varias bases de datos. Soporta gran tráfico de usuarios.

UCI: Universidad de las Ciencias Informáticas.

UML: Unified Modeling Language o Lenguaje Unificado de Modelado. Es usado en la documentación, descripción y construcción del software. Está reconocida en la industria del software como un estándar.

Usuario: Individuo u organización que interactúa con un sistema.

XP: Denominada eXtreme Programming o Programación Extrema, XP es una de las MDS más exitosas en la actualidad.