

**Instituto Tecnológico y de Estudios
Superiores de Monterrey
Campus Guadalajara**

**Escuela de Graduados en Ingeniería y
Arquitectura (EGIA)**

Maestría en Ciencias Computacionales

áéíóúÁÉÍÓÚñÑüÜ

AUTOR: Marco Antonio Rangel Bocardo

ASESORES: Dr. Óscar Mondragón

Guadalajara (Jal), 11 de Septiembre de 2011

Dedicatoria

*A mis padres,
a mis hermanos,
y a mis entrañables
amigos*

Agradecimientos

Quizá no estaba en plenitud de mis sentidos en el momento que decidí iniciar una Maestría en Ciencias de la Computación. Definitivamente se necesita un poco de locura, y un tanto más de agallas para embarcarse en una maestría con enfoque científico, en la que tengas que crear y defender una tesis para obtener el grado. Esto se pone un poco más interesante si le agregamos el hecho de tener un trabajo de tiempo completo y otro de medio tiempo mientras se cursa la maestría. Cuando se ve esta perspectiva de una forma más general, se comprende que esto no lo hubiera logrado yo solo, y que, de alguna manera u otra, se ha llegado al objetivo gracias al apoyo de varias personas a las que quisiera agradecer personalmente y en esta dedicatoria.

Primero que nada quiero agradecer al Doctor Óscar Mondragón, el director de esta tesis, por todo el apoyo que me ha brindado durante este trabajo de un año. Él decidió desinteresadamente apoyarme, y me ayudó a transformar una serie de ideas en un producto completo y palpable. Con sus conocimientos y experiencias personales y profesionales, fue una gran guía durante toda esta travesía.

También es muy importante mencionar a mis compañeros que elaboraron junto conmigo el sistema que es la base del estudio. Eduardo Campos y Humberto García, aparte de ser dos estudiantes sobresalientes, son aún mejores como personas, siendo destacados por tener valores como la responsabilidad, la honestidad, el trabajo duro y un sentido de compromiso como pocas personas en nuestro país.

Quisiera agradecer también a mis padres y hermanos, Antonio Rangel, Patricia Bocardó, Julia Rangel, Bryan Rangel y Ángel Rangel, por todo el apoyo que me brindan en cada uno de mis proyectos, y aunque no hayan cooperado de forma directa con el trabajo, su apoyo moral siempre ha sido muy importante para lograr lo que me propongo.

Cuento con la suerte de tener muchos y excepcionales amigos, así que no quería perder la oportunidad de mencionarlos, ya que ellos hacen que mi vida sea más plena y esté llena de experiencias divertidas. Agradezco mucho a: Guille Suro, Naiv Soto, Jafet Rodríguez,

Humberto García, Héctor Solís, Julián Hernández, Leonel Haro y Joaquín Soto; por todos los buenos momentos, las risas y las aventuras que hemos vivido.

Finalmente quisiera agradecer de forma especial a Alejandro Vázquez. Él nos brindó apoyo técnico y conocimientos durante la elaboración del sistema, todo de forma desinteresada, sin recibir nada a cambio, y siempre con una actitud de ayuda y servicio.

Resumen

Este trabajo de Tesis presenta el sistema de administración que se llamará "Bug Manager"(BM). Con este sistema se demuestra como la implementación de conceptos simples y claves de calidad dentro del proceso de desarrollo de software en empresas medianas y pequeñas, ayudan a mejorar sustancialmente los tiempos de entrega, la calidad final del producto y la satisfacción de los clientes. Algunos de estos conceptos son:

- Registro de actividades con su esfuerzo y tamaño dedicados;
- Registro y seguimiento de defectos encontrados;
- Realización de revisiones de código;
- Medición de la productividad personal y global de la empresa.

Con estos conceptos en mente, se propone la creación del BM, el cual será utilizado por cualquier empresa que busque implementar técnicas de control de calidad en su proceso de desarrollo de software. Este producto va enfocado inicialmente a pequeñas y medianas empresas, principalmente mexicanas. Esto último en virtud de que es el mercado preferente al cual se pretende llegar.

El BM permitirá que las empresas tener control sobre el avance y la calidad de sus proyectos, por medio de las siguientes estrategias:

- Guía en la elaboración del plan de calidad.
- Definición del ciclo de vida y actividades de desarrollo.
- Registro y seguimiento de actividades de aseguramiento de calidad.
- Registro y seguimiento de defectos.
- Generación de estadísticas personales, por proyecto, por equipo y por empresa.

Finalmente, durante el desarrollo del BM se pondrán en práctica las actividades de calidad mencionadas, y se hará un análisis del costo de la calidad para comprobar la efectividad de estas actividades en la mejora del proceso de desarrollo de software.

Contenido

Dedicatoria	I
Agradecimientos	II
Resumen	IV
Lista de Tablas	IX
Lista de Figuras	X
1. Introducción	1
1.1. Antecedentes	1
1.2. Planteamiento del Problema	2
1.3. Propuesta de Solución	3
1.4. Objetivos	5
1.5. Alcances	5
1.6. Contribuciones	6
2. Marco Teórico	7
2.1. Nociones Básicas de Calidad de Software	7
2.1.1. Definición de Calidad de Software	7
2.2. Enfoques de la Calidad de Software	8
2.2.1. Adherencia a Procesos	8
2.2.2. Pruebas de Software	9
2.2.3. Revisiones de Productos de Trabajo	11
2.3. Importancia de la Calidad en el Desarrollo de Software	11

2.3.1.	El software en las organizaciones	12
2.3.2.	¿Por qué los proyectos de software fallan?	12
2.3.3.	La Calidad es Negocio	13
2.3.4.	Introduciendo la Calidad a las Organizaciones	14
2.3.5.	Beneficios del Trabajo de Calidad	16
2.4.	El Proceso Personal del Software	16
2.4.1.	¿Qué es la calidad de software?	17
2.4.2.	La Economía de la Calidad de Software	18
2.4.3.	Tipos de Defectos	19
2.4.4.	Métricas de Calidad	19
2.4.5.	Prácticas para la Mejora de Calidad del PSP	21
2.4.6.	Prevención de Defectos	22
2.4.7.	Técnicas de Detección de Defectos	22
2.4.8.	¿Por qué Revisar los Programas?	24
2.4.9.	Principios de la Revisión	24
2.4.10.	Lista de Chequeo de Revisión de Código	25
2.4.11.	Evalando las Revisiones Personales	27
2.4.12.	Efectividad de la Revisión	28
2.4.13.	Diseño de Software	28
2.4.14.	El Proceso de Diseño	29
2.4.15.	Niveles de Diseño	30
2.4.16.	Calidad y el Diseño	30
2.4.17.	Verificación de Diseño	31
2.5.	Defectos de Software	34
2.5.1.	Clasificación Ortogonal de Defectos	35
2.5.2.	Inyección de Defectos y Eficiencia en la Remoción de Defectos	37
2.5.3.	El Costo Real de los Defectos de Software	38
2.6.	Costo de la Calidad de Software	39
2.6.1.	De las Pruebas a la Prevención	41
2.6.2.	Análisis del Costo de la Calidad	42

2.6.3. Análisis del Retorno de Inversión	43
3. Desarrollo del Trabajo	45
3.1. Concepto de Operaciones	45
3.1.1. Objetivos	45
3.1.2. Alcances	46
3.1.3. Módulos del Sistema	46
3.1.4. Tipos de Usuario	47
3.1.5. Impacto	49
3.1.6. Limitaciones	50
3.2. Diseño del Sistema	51
3.2.1. Arquitectura	51
3.2.2. Base de Datos	53
3.3. Funcionalidades del BM	54
3.3.1. Administración de Usuarios	55
3.3.2. Administración de Recursos	55
3.3.3. Administración de Proyectos	55
3.3.4. Ciclo de Vida de Proyectos	56
3.3.5. Administración y Seguimiento de Actividades	58
3.3.6. Administración y Seguimiento de Defectos	62
Bibliografía	65

Lista de Tablas

2.1. Número de Defectos por Nivel de CMM	17
2.2. Costo de Encontrar y Corregir Defectos	18
2.3. Tipos de Defectos por Mil Líneas de Código	19
2.4. Métricas del COQ	20
2.5. Métricas del PQI	21
2.6. Clasificación de Defectos del PSP	26
2.7. Lista de Chequeo para la Revisión de Código de PSP	26
2.8. Potencial de Inyección Defectos Promedio	37
2.9. Efectividad en la Remoción de Defectos	37
2.10. Análisis del ROI	44

Lista de Figuras

2.1. Costo de la Remoción de Defectos por Fase	38
2.2. Costo de la Calidad de Software	39
2.3. El costo de la alta confiabilidad	40
2.4. Modelo de la Calidad de Software	41
2.5. CoQ por nivel de CMM	43
3.1. Módulos del BM	47
3.2. Tipos de usuario del BM	48
3.3. Arquitectura del BM	52
3.4. Base de Datos del BM	53
3.5. Ciclo de Vida Default	57
3.6. Edición del Ciclo de Vida	58
3.7. Creación de Actividades	61
3.8. Seguimiento de Actividades	62

CAPÍTULO 1

Introducción

1.1 Antecedentes

Consultores de modelos de calidad [CITAR] de las empresas mexicanas de desarrollo de software han identificado un común denominador en las empresas pequeñas y medianas: Una pobre o nula administración de la calidad en el proceso de desarrollo.

En este tipo de situación, las empresas suelen asumir que las prácticas de calidad agregan trabajo extra, haciendo más lento y complicado el proceso de desarrollo. Esto provoca el atraso en los calendarios y la entrega tardía de los productos respectivos..

Estas decisiones provocan exactamente el efecto contrario. Al no tener un apropiado control de la calidad de sus productos, se ven envueltos en las siguientes situaciones:

- Cuando se llega a la fase de pruebas, el producto está plagado de defectos; lo que ocasiona que la fase tome la mitad del tiempo total de desarrollo, haciendo para muchos tortuosa esta etapa;
- Una vez que el sistema sale a producción, no está garantizado que el producto no tiene defectos. Muchos de estos fueron generados al momento de hacer las correcciones, o simplemente no se encontraron;
- Cuando los usuarios encuentran defectos en el producto final, lo usual es hacer la corrección de estos defectos. Corregir un evento en etapa de pruebas cuesta normal-

mente diez veces más de lo que costaría en la fase de codificación; tanto como corregir un defecto en la fase mantenimiento cuesta cien veces más que hacerlo en la fase de codificación [1]. Ambas generan altos costos de mantenimiento los cuales suelen ser absorbidos por la empresa que desarrolló el sistema.

- Peor aún que el incremento de los costos de mantenimiento, el cliente tiene un producto defectuoso, el cual no le permite realizar las actividades requeridas, generando desconfianza en la empresa de desarrollo, además una mala imagen y pérdidas de clientes en el futuro.

Todas estas situaciones pueden ser evitadas con una correcta administración de la calidad en el proceso de desarrollo de software.

Lo que muchas empresas no tienen en cuenta es que la calidad en el desarrollo debería ser la prioridad en el proceso de elaboración de productos de software. La calidad ayuda a que los productos en desarrollo sean predecibles en tamaño y calendario, fáciles de dar seguimiento y bajo costo de mantenimiento[2].

En vez de hacer más largo, complicado y costoso el proceso de desarrollo, la administración de la calidad recorta los tiempos de desarrollo, reduciendo considerablemente la fase de pruebas, disminuyendo el ciclo de vida del proyecto. La reducción del tiempo total de desarrollo se traduce en ahorro de costo en el proyecto. Si a lo anterior agregamos un menor número de defectos en las etapas de pruebas y producción, el cliente tendrá un producto de calidad, y la empresa de desarrollo mejorará imagen y su perspectiva de mercados a futuro.

1.2 Planteamiento del Problema

La problemática consiste en que las empresas pequeñas y medianas de desarrollo de software no elaboran productos de calidad. Esto ocasiona que al construir los productos de software la fase de pruebas tome hasta la mitad del tiempo total del proyecto; que el proyecto se salga de calendario; y que al entregar el producto al cliente este aún tenga defectos y este no quede satisfecho con el trabajo realizado. Esta problemática se origina por las siguientes situaciones:

- Compromisos poco realistas con el cliente;
- Una pobre administración del proyecto;
- No tener procesos definidos de desarrollo;
- No tener un plan de administración de la calidad;
- Seguimiento inadecuado de los defectos encontrados.

1.3 Propuesta de Solución

A partir de los antecedentes y la problemática descrita en el punto anterior, se propone la creación de una herramienta de administración de la calidad en el software bautizada Bug Manager (por sus siglas en inglés BM), que ayude a las pequeñas y medianas empresas a la implementación y seguimiento de un plan de calidad, así como las actividades que se requieran realizar. Esta herramienta ataca el nicho de estas pequeñas y medianas empresas las cuales no cuentan con procesos de calidad ni con presupuestos para la compra de herramientas.

El BM ayudará a las empresas a:

- Establecer un ciclo de desarrollo.
- Elaborar un plan de calidad estableciendo objetivos y técnicas de detección de defectos para cada fase del desarrollo.
- Ayudar con plantillas que sirvan como guías de las técnicas de detección de defectos.
- Dar un seguimiento apropiado a los defectos encontrados durante el desarrollo del sistema.
- Generación de estadísticas y reportes los cuales mostrarán información valiosa acerca del desarrollo como: Productividad, Densidad de Defectos, Retorno de Inversión de las Actividades de Calidad, entre otras. Estas proporcionarán a las empresas información importante acerca de su proceso de desarrollo, mostrando cuáles son sus áreas fuertes y en cuáles hay una oportunidad de mejora.

También se pretende que el BM genere una actitud de calidad total y mejora continua en las empresas que lo utilicen, y lograr un cambio cultural evolutivo:

- Promoviendo una cultura de calidad personal en el programador en lugar de una cultura de calidad asignada a grupos organizacionales ajenos al desarrollo (pruebas, adherencia a procesos).
- Estableciendo una meta en el programador/grupo de desarrollo de cero defectos en pruebas de unidad contra número de componentes programados por hora.
- Promoviendo la prevención de defectos en lugar de la búsqueda de defectos durante las pruebas.
- Enfocando el esfuerzo de técnicas de detección de defectos al inicio del ciclo de vida en lugar de crecer los grupos dedicados a las pruebas al final del ciclo de vida.
- En resumen, promover un compromiso personal a la calidad del desarrollo de software y a las actividades asociadas para su mejora continua.

En la actualidad existen herramientas y programas de software que realizan tareas similares a las que realizará el sistema propuesto. Principalmente estos sistemas se dedican al registro y rastreo de defectos, así como al registro de las actividades realizadas dentro del ciclo de desarrollo, como una especie de bitácora. En general, consideramos que estos sistemas atacan una parte del problema y comúnmente carecen de la funcionalidad necesaria para agregar verdadero valor al proceso y método de desarrollo, debido a que se centran ya sea en el seguimiento de actividades o en el registro de defectos, pero no conjuntan ambas vistas de la problemática, aparte de que no generan estadísticas acerca de la información recabada.

El sistema propuesto pretende, al igual que las otras herramientas dar un seguimiento apropiado a los defectos, así como el establecimiento y la guía de un plan de calidad, finalmente generando estadísticas y reportes de todos los datos recabados.

1.4 Objetivos

Los objetivos del Trabajo de Tesis son:

- Realizar una investigación y análisis de los factores que determinan la calidad en el proceso de desarrollo de software, y demostrar el papel clave de la calidad en el desarrollo de software.
- Realizar una investigación y análisis del Costo de la Calidad en el proceso de desarrollo de software. Comparación del costo de la calidad contra el costo de la no calidad. Análisis del retorno de inversión de las distintas técnicas y prácticas de calidad.
- Realizar la propuesta de la herramienta BM, en las partes que conciernen al Costo de la Calidad.
- Construcción de la herramienta BM, en las partes que conciernen al Costo de la Calidad.
- Análisis de los resultados obtenidos en el proceso de construcción de la herramienta BM, que incluye el análisis de Costo de la Calidad contra Costo de la No Calidad, y análisis del Retorno de Inversión de las actividades y prácticas implementadas.
- Que la herramienta BM tenga las siguientes características mínimas:
 - Generar estadísticas y métricas de valor para la empresa y el personal en base a la información proporcionada por los usuarios del sistema.
 - Dar una guía en los procedimientos principales de aseguramiento de la calidad.
 - Optimizar y hacer más eficiente el proceso de desarrollo de software promoviendo actividades de prevención de defectos y análisis de datos.

1.5 Alcances

Los alcances del Trabajo de Tesis son:

- Investigación del Costo de la Calidad en el proceso de desarrollo de software.
- Propuesta y construcción de la herramienta BM, en las partes que conciernen el Costo de la Calidad.

1.6 Contribuciones

Las contribuciones del trabajo de tesis son:

- Proporcionar una herramienta flexible y efectiva para realizar las diferentes actividades de calidad, que permita eventualmente cambiar la manera en la que se elaboran este tipo de herramientas hasta el día de hoy.
- Que la herramienta elaborada colabore en la mejora continua de los productos de software desarrollados, por las diferentes empresas que adopten la herramienta como parte de su ciclo de desarrollo.
- Colaborar con la industria mexicana y latinoamericana de desarrollo de software, especialmente en las pequeñas y medianas empresas, a generar una cultura de calidad que ayudará a atraer más proyectos a la industria, generando así una mejora económica en la región.

CAPÍTULO 2

Marco Teórico

2.1 Nociones Básicas de Calidad de Software

En esta sección se darán las nociones básicas que se deben de conocer en el tema de la Calidad de Software antes de poder hablar de temas más específicos.

2.1.1 Definición de Calidad de Software

El concepto de Calidad de Software ha sido definido de varias formas por distintos autores. A continuación se proveen algunas de las definiciones más destacables:

- “Calidad significa cumplir con los requerimientos” - [3];
- “Calidad consiste en las características de los productos que cubren las necesidades de los clientes produciendo satisfacción gracias al producto” - [4];
- “Calidad consiste en la libertad de deficiencias” - [4];
- “Calidad de Software es: El grado en que un sistema, componente o proceso cumple con los requerimientos especificados” - IEEE 1991;
- “Calidad de Software es: El grado en que un sistema, componente o proceso cumple con las necesidades o expectativas del usuario” - IEEE 1991;
- “Como la belleza, todos tienen su idea de que es la calidad” - [5].

Tomando en cuenta estas definiciones, hay autores que se inclinan por definir Calidad de Software en relación al cumplimiento de requerimientos, y otros que prefieren relacionar la calidad con la satisfacción final del cliente. Finalmente las empresas deben de buscar un balance apropiado entre ambos enfoques para lograr el éxito en su labor.

2.2 Enfoques de la Calidad de Software

Existen tres principales enfoques bajo los cuales la industria busca mejorar la calidad en el desarrollo de software. Estos son: Adherencia a procesos, elaboración de pruebas y revisiones de producto.

2.2.1 Adherencia a Procesos

Un enfoque que se tiene para dar calidad a los sistemas de software se hace mediante la definición y adherencia a procesos. La idea de este enfoque es que las organizaciones tengan procesos bien definidos, y los sigan religiosamente. Esto con el objetivo de siempre poder obtener resultados similares en sus diferentes procesos.

Dos grandes ejemplos de este enfoque son el ISO9000 el modelo de capacidad y madurez integrado (Por sus siglas en Inglés, CMMI).

ISO9000 es una familia de estándares relacionados a la administración de la calidad de los sistemas, y están diseñados para ayudar a las organizaciones a asegurarse que cumplen con las necesidades de los clientes y las demás partes interesadas. Los estándares que componen al ISO9000 son [6]:

- *ISO9000*. Es la introducción al ISO 9000, se refiere a la selección y uso de los ISO9001-9004;
- *ISO9001*. Modelo para asegurar la calidad en el diseño, desarrollo, producción e instalación en las organizaciones;
- *ISO9002*. Modelo para asegurar la calidad en la producción, instalación y servicio. Está basado en el ISO9001 pero también incluye la parte de creación de nuevos productos;

- *ISO9003*. Modelo para asegurar la calidad en las inspecciones finales y el proceso de pruebas;
- *ISO9004*. Modelo para asegurar la calidad a través de situaciones preventivas.

CMMI es un modelo de referencia que guía iniciativas de mejora de procesos que tiene como meta ayudar a las empresas a mejorar su desempeño. Este modelo contiene los elementos esenciales de los procesos efectivos, y describe un proceso evolutivo de mejora tanto para mejorar procesos inmaduros como para mejorar procesos maduros. Cuenta con 5 niveles de madurez que son los siguientes[7]:

- *Nivel 1 (Inicial)*. Procesos impredecibles, poco controlados y que reaccionan a las situaciones;
- *Nivel 2 (Administrado)*. Proceso caracterizado por estar orientado a nivel proyecto, normalmente es reactivo a las situaciones;
- *Nivel 3 (Definido)*. La organización tiene procesos definidos y es proactiva. Los proyectos se rigen mediante los procesos de la organización;
- *Nivel 4 (Administrado Cuantitativamente)*. Los procesos de la organización son controlados y medidos cuantitativamente;
- *Nivel 5 (Optimización)*. Se enfoca en la mejora continua de procesos.

2.2.2 Pruebas de Software

Un segundo enfoque es aquel que está basado en la realización de pruebas para asegurar la calidad del sistema. Las pruebas de software incluyen un proceso de verificación y validación de un sistema de software. Estas actividades se realizan con el objetivo de encontrar defectos en un sistema de software. Un defecto es una diferencia entre el resultado esperado y el resultado obtenido.

Las pruebas de software son realizadas para encontrar los defectos en el sistema de software antes de que estos lleguen al usuario final. Existen distintos tipos de pruebas de software, los más comunes son [8]:

- *Pruebas Unitarias.* Son pruebas ejecutadas a una unidad de sistema. Dependiendo del tipo de proyecto, una unidad puede ser considerada, un método dentro de alguna clase, un archivo de código fuente completo entre otras. En estas pruebas solo se asegura la funcionalidad asociada a la unidad.
- *Pruebas de Sistema.* Es ejecutar todas las pruebas unitarias en un escenario real. Se refieren a utilizar el sistema bajo condiciones normales donde las unidades interactúan entre sí.
- *Pruebas de Estrés.* Es ejecutar pruebas de sistema pero bajo condiciones de estrés, con una carga de trabajo muy alta y fuera de lo normal.
- *Pruebas de Aceptación de Usuario.* También conocidas como pruebas Beta, tratan de darle una versión funcional, aunque no propiamente final, del sistema al usuario final, para que estos se aseguren de que el sistema hace lo que ellos necesitan y lo haga de forma correcta.
- *Pruebas de Regresión.* Estas pruebas se ejecutan cuando se realiza la corrección de algún defecto. Todos los componentes o unidades que son afectados por este cambio tienen que ser probados de nuevo para asegurarse de que la corrección del defecto original no introdujo nuevos defectos.

Considerar los tipos de prueba como la única alternativa para remover defectos es inadecuado, costoso, poco efectivo y conlleva a perder el control del proyecto en la fase de pruebas. El enfoque de este Trabajo de Tesis es justamente evitar hacer pruebas extensivas. Como se mencionó en la sección 1.2 del presente Trabajo de Tesis, gran parte de la problemática actual se debe a que los productos del proceso de desarrollo de software tienen una alta densidad de defectos por lo que llegan con una calidad muy baja a la etapa de pruebas, que a las organizaciones de desarrollo normalmente les toma la mitad del tiempo total del proyecto terminar esta etapa [2]. El Trabajo de Tesis se enfoca en la revisión y prevención de defectos durante el ciclo de vida, con el último objetivo de que los productos del proceso de desarrollo de software lleguen libres de defectos a la etapa de pruebas.

2.2.3 Revisiones de Productos de Trabajo

Las revisiones de los productos de trabajo son evaluaciones y revisiones de los productos creados durante el ciclo de vida de desarrollo de software ya sea de forma personal, o por parte de uno o más compañeros del equipo de desarrollo calificados para hacerlo. Este es el enfoque del Trabajo de Tesis.

La idea principal detrás de las revisiones es detectar los defectos lo antes posible en el proceso de desarrollo de software. Esto evita que los errores permanezcan durante el proceso de desarrollo, donde se multiplican y se va haciendo más difícil encontrarlos y corregirlos mientras avanza el proyecto[9].

Los tipos de revisiones de productos más comunes son: Revisiones personales y entre colegas; caminatas e inspecciones. En la subsección 2.4.7 del presente Trabajo de Tesis se esbozarán distintas estrategias para promover la cultura de la revisión y la prevención.

2.3 Importancia de la Calidad en el Desarrollo de Software

El software es una tecnología sorprendente. Es un producto totalmente intelectual, aparte puede ser distribuido mundialmente en segundos, no se deteriora con el tiempo y es la forma más económica y sencilla de implementar casi cualquier función compleja.

En todos los campos de la ingeniería y la ciencia, más de la mitad del tiempo de cualquier profesional lo utiliza desarrollando, mejorando, manteniendo o usando un sistema de software. Es sin duda uno de los negocios más grandes e importantes de la actualidad[2].

Tomando en cuenta lo anterior, muchos ejecutivos de varias organizaciones vieron las oportunidades de negocio en el software, sin embargo implementar un departamento de software efectivo no es una tarea trivial, y la calidad en los productos que se desarrollen es clave para el éxito.

2.3.1 El software en las organizaciones

Para administrar una organización o un departamento dentro de una organización dedicado al software se necesita tener en cuenta los siguientes principios de administración[2]:

1. Reconocer la importancia del software en el negocio;
2. La calidad en los productos de software debe de ser la principal prioridad. La calidad es una elección con consecuencias económicas, si no se paga por la calidad al principio del desarrollo, se pagará una cantidad mucho más alta después. Para realmente lograr que los proyectos cumplan con los calendarios y los costos, el trabajo tiene que realizarse de forma correcta desde el principio;
3. El software de calidad se desarrolla por personas disciplinadas y motivadas. Si los profesionales del software no están convencidos y entrenados en métodos de calidad, no van a seguir las prácticas requeridas y no producirán software de calidad.

2.3.2 ¿Por qué los proyectos de software fallan?

La razón principal por la que los proyectos fallan es administración inadecuada. Una buena administración requiere dos cosas: Estar enfocados en la calidad, e ingenieros motivados que realicen un trabajo disciplinado. Las causas más comunes por las cuales los proyectos fallan son[2]:

- *Calendarios poco realistas.* Cuando un proyecto de software inicia con calendarios poco realistas, el proyecto será entregado después de lo que se hubiera hecho con un calendario realista. Esto es porque los ingenieros comienzan a hacer trabajo rápido y de poca calidad para alcanzar el calendario. El resultado de esto son productos de baja calidad, los cuales están llenos de defectos, lo que se traduce a una extensa etapa de pruebas;
- *Equipo inadecuado.* La única forma de terminar un proyecto de software de manera rápida y efectiva es asignar el número adecuado de personas y protegerlas de interrupciones y distracciones;

- *Cambio de requerimientos.* Es normal que los requerimientos cambien en las fases iniciales del proyecto, sin embargo llega un punto que esta situación es muy perjudicial. Se tiene que identificar este punto para evitar pérdidas de dinero y cambios que afecten en sobremanera el trabajo que se está haciendo;
- *Baja calidad en el trabajo realizado.* Si el trabajo que se realiza es de baja calidad, va a ocasionar que la fase de pruebas sea muy larga y se gaste mucho tiempo arreglando defectos durante el proceso de desarrollo;
- *Creer en la magia.* No existe la bala de plata. Muchas veces se cree que con una nueva tecnología o forma de desarrollo resolverá todos los problemas, y lo que ocurre generalmente es que pueden llevar a serios problemas.

El costo de una baja calidad de software es difícil de ver hasta el final del proyecto. Comúnmente, la baja calidad le dará problemas aún a los usuarios finales ya que el proyecto se haya terminado. Los errores más comunes son: líderes de proyecto que toman compromisos irresponsables y no insisten en que el trabajo se haga de forma correcta. Todos los problemas mencionados anteriormente se pudieron evitar si la administración hubiera insistido en planear correctamente el trabajo y realizarlo de forma disciplinada.

2.3.3 La Calidad es Negocio

Hay tres razones fundamentales para insistir en que la Calidad de Software tiene que ser medida y administrada[2]:

1. La baja calidad en el software puede causar daños severos a la propiedad, y en algunos casos puede matar personas;
2. El trabajo de calidad ahorra tiempo y dinero;
3. Si la gerencia y el líder de proyecto no insisten en la administración de la calidad de software, nadie más lo hará.

Varios estudios han demostrado que incluso los ingenieros experimentados inyectan un defecto cada diez líneas de código[2]. Esto no significa que los programadores sean incompetentes, simplemente son humanos.

El costo de encontrar y corregir los defectos se incrementa en cada fase del proceso de desarrollo de software. Entre más permanezca un defecto en el sistema, será más difícil removerlo, a demás de que se multiplica.

Existen varias técnicas para remover defectos antes de llegar a la fase de pruebas. Estas técnicas fueron serán descritas a mayor profundidad en la sección 2.4.7 . Una de estas técnicas por ejemplo, es la revisión de código, en las que se lee y revisa el código producido en busca de defectos. Estas actividades suelen tomar una pequeña cantidad de tiempo, y cada defecto encontrado ahorra una gran cantidad de tiempo en la fase de pruebas.

A diferencia de las revisiones de código, la fase de pruebas es una actividad de calidad mucho más larga. La razón de esto es que en las pruebas de software solo se revela los síntomas del defecto, mientras que en las revisiones se encuentra directamente el defecto.

La mejor manera de remover defectos de software es realizando revisiones de código, ya que estas son más baratas y más efectivas que las pruebas de software. Adicionalmente, el ingeniero que desarrolló el programa es el más indicado para encontrar y corregir sus propios defectos.

Aun conociendo la efectividad de las revisiones de código, muchas organizaciones no las utilizan. La razón es que para realizar revisiones de código efectivas se requieren métodos disciplinados.

Si los proyectos tienen la mayor parte de su tiempo dedicado a la fase de pruebas de software, va a ser casi imposible planearlo y darle seguimiento. Si se desea que los planes tengan compromisos precisos y realizables, se debe insistir en realizar un trabajo de calidad incluyendo actividades de detección de defectos.

2.3.4 Introduciendo la Calidad a las Organizaciones

Una organización exitosa siempre busca mejorar sus procesos por medio de reducir el tiempo que toman, aumentar su calidad y reducir sus costos. Un principio básico de

la administración que nos permite llegar a este objetivo es: Lo que se mide puede ser administrado y lo que es administrado puede ser realizado. Al contrario, lo que no es medido comúnmente se ignora.

Para que una organización pueda construir software utilizando procesos de calidad que resulten en sistemas de alta calidad, debe de utilizar la administración racional. El principio de administración racional es confiar en que los miembros de un equipo de desarrollo de software son profesionales preocupados por el éxito del proyecto. En la administración racional se necesita crear planes, seguir estos planes, y corregir los problemas antes de que se salgan de control, también hace un énfasis especial en el calendario, la calidad y los costos. Los cuatro elementos principales de la administración racional son[2]:

1. Establecer metas agresivas a largo plazo, pero descomponerlas en metas realísticas y medibles a corto plazo;
2. Insistir en la creación de planes e insistir que estos planes sean creados por las personas que harán el trabajo. Los planes tienen que ser detallados y completos, y tienen que ser revisados a conciencia en busca de omisiones y otros errores por los líderes de proyecto y la gerencia;
3. Los planes de trabajo deben de ser utilizados al momento de realizar el trabajo. Esto permite realizar balanceo de cargas y conocer el estado actual del proyecto. Todos los productos deben de tener alta calidad, si no es así eventualmente tendrán que ser corregidos y tendrán un costo extra.
4. Se tienen que utilizar datos e información. Si se utiliza de forma objetiva los datos y la información del desempeño del equipo, se demuestra la confianza que se tiene en el equipo y la disposición que existe para escuchar sus problemas, planes e ideas;
5. La reducción de costos se logra por medio de maximizar el tiempo de trabajo de los ingenieros, y esto se hace planeando el tiempo que tomarán las actividades y midiendo el tiempo que realmente tomaron.
6. Dar seguimiento al trabajo realizado y utilizar la información actual para anticipar y resolver problemas futuros. Los calendarios y planes se salen de control día a día.

En resumen, para que una organización pueda mejorar su calidad las diferentes partes deben de cumplir con lo propuesto anteriormente. Los ingenieros deben de utilizar las prácticas de calidad, así como planear y dar seguimiento al trabajo realizado y finalmente medir y administrar la calidad del producto. Los líderes o administradores de proyecto deben iniciar y mantener este nuevo comportamiento, así como monitorear los planes y balancear efectivamente la carga de trabajo. La gerencia o líderes de la organización deben de proveer un ambiente disciplinado y atractivo de trabajo, establecer las metas a largo plazo y dar seguimiento al trabajo realizado. Cuando el trabajo es planeado, medido y monitoreado, se puede analizar el desempeño del proyecto y del negocio.

El BM proveerá a las organizaciones con la infraestructura básica para la implementación de una administración racional. Permitirá establecer planes, dar de alta tareas con sus tiempos planificados, darles seguimiento y registrar sus tiempos reales. Más detalles de esto se darán en la sección 3.3.5.

2.3.5 Beneficios del Trabajo de Calidad

Cuando se utilizan correctamente métodos disciplinados para planear, dar seguimiento y administrar, el trabajo será de alta calidad y será terminado dentro de calendario. Más importante aún, tendremos una noción bastante precisa de en qué parte del proceso de desarrollo se encuentra el proyecto, y una predicción fiable de cuándo se va a terminar.

El tiempo que se requiere para escribir programas de alta calidad es el mismo que se necesita para escribir programas de baja calidad. Los productos de calidad reducen el tiempo desarrollo. Entre menos defectos existan, el costo será menor y los estimados serán más efectivos. También el tiempo que se requiera para terminar la etapa de pruebas será mucho menor.

2.4 El Proceso Personal del Software

El Proceso Personal del Software (por sus siglas en Inglés PSP) es una metodología de mejora continua en el desarrollo de software propuesta por Watts Humphrey en su libro “PSP A Self-Improvement Process for Software Engineers”[1]. El PSP ayuda a controlar,

administrar y mejorar los procesos para desarrollar software. Provee una infraestructura de guías, plantillas y procedimientos para: desarrollar software, dar seguimiento a los defectos y administrar la calidad. Esto hace que los productos de trabajo sean más predecibles y eficientes, que se elaboren mejores planes, dar un seguimiento adecuado al desempeño y medir la calidad de los productos elaborados.

El BM toma varios conceptos de esta metodología para alcanzar los objetivos que propone. En esta sección serán descritos los conceptos de los que hace uso el BM.

2.4.1 ¿Qué es la calidad de software?

Los métodos tradicionales para asegurar la calidad del software en desarrollo son: Inspección de requerimientos, inspección de diseño, compilación, inspección de código y pruebas. La tabla 2.1 muestra los niveles de defectos promedio por las organizaciones según su nivel de CMM[1].

Nivel de CMM	Defectos/Mil de Líneas de Código
1	7.5
2	6.24
3	4.73
4	2.228
5	1

Tabla 2.1: Número de Defectos por Nivel de CMM

Para mejorar la calidad de productos de software, aparte de utilizar los métodos tradicionales de calidad, se debe de medir y dar seguimiento al trabajo personal. Sí el objetivo es tener un sistema de software de alta calidad, cada parte del sistema debe de ser de alta calidad. La estrategia de PSP es administrar los defectos contenidos en todas las partes del sistema. Con partes de alta calidad, el proceso de desarrollo de software se puede escalar sin perder productividad.

La definición de calidad debería de basarse en las necesidades de los usuarios[1]. Los usuarios deben de contar con un producto funcional. Si el producto tiene muchos defectos este no se desempeñará con una consistencia razonable. Sin embargo esto no significa que los defectos son siempre la principal prioridad. Después de los defectos se tienen características de calidad como: Desempeño, seguridad, usabilidad, compatibilidad, funcionalidad,

confiabilidad entre otras. Normalmente se gasta un tiempo excesivo en la corrección de defectos, dejando muy poco tiempo dedicado a las otras características mencionadas anteriormente.

Aunque los defectos son solo una parte de la calidad del producto de software, son el principal objetivo del PSP, ya que la administración efectiva de estos es esencial para la administración de costo, calendario y los demás aspectos de la calidad de producto.

2.4.2 La Economía de la Calidad de Software

La calidad de software es un problema de económico. Cada prueba realizada cuesta dinero y toma tiempo. Entre más tiempo el defecto permanezca en el producto, el impacto es más alto. Por ejemplo encontrar un problema de requerimientos cuando el cliente ya está utilizando el producto puede ser demasiado costoso; en cambio, encontrar el defecto durante una revisión de código será mucho menos costoso. El objetivo, entonces, es remover los defectos lo antes posible en el proceso de desarrollo. Esto se logra haciendo revisiones e inspecciones de cada producto de trabajo lo más pronto posible, antes de que se haya terminado de construir el producto de software.

El proceso de pruebas puede ser muy efectivo para identificar problemas de desempeño, usabilidad y problemas operacionales, pero no es tan efectivo removiendo grandes cantidades de defectos. Los datos de estudios realizados demuestran que mientras más tarde en el proceso de desarrollo sea encontrado un defecto, es más difícil encontrarlo y removerlo[1]. La tabla 2.2 nos muestra cuanto cuesta encontrar y corregir defectos en Inspecciones, Pruebas y Mantenimiento según distintos estudios:

Referencia	Inspección	Pruebas	Uso
IBM (Remus and Ziles 1979) [10]	1	4.1 veces la inspección	
JPL (Bush 1990)[11]	\$90 - \$120	\$10,000	
Ackerman et al. 1989[12]	1 hora	2-20 horas	
Ragland 1992[13]		20 horas	
Russell 1991[14]	1 hora	2-4 horas	
Shooman and Bolsky 1975[15]	0.6 horas	3.05 horas	
Weller 1993[16]	0.7 horas	6 horas	

Tabla 2.2: Costo de Encontrar y Corregir Defectos

2.4.3 Tipos de Defectos

Los tipos de defectos más comunes son: Errores en requerimientos, errores de diseño, errores de codificación, errores de documentación, correcciones defectuosas. La tabla 2.3 presenta el número de defectos cada mil líneas de código según el tipo de defecto[1].

Tipos de Defecto	Defectos/Mil de Líneas de Código
Requerimientos	2.3
Diseño	1.9
Codificación	0.9
Documentación	1.2
Correcciones Defectuosas	1.2
Total	7.5

Tabla 2.3: Tipos de Defectos por Mil Líneas de Código

2.4.4 Métricas de Calidad

Para mejorar la calidad, se debe de medir la calidad[1]. El PSP propone el uso de las siguientes métricas para medir la calidad de las organizaciones de software: La efectividad de la técnica de detección de defectos (Yield de Calidad), el Costo de la Calidad (Por sus siglas en Inglés, COQ), la Tasa de Revisiones, la Calidad de Radio de Fases y el Índice de Calidad de Proceso.

El Yield mide la eficiencia de cada fase en la detección de defectos. El Yield de una fase es el porcentaje de defectos de producto totales que son removidos en esa fase. El Yield de proceso es el porcentaje de defectos que son removidos antes de la primera compilación o pruebas de unidad. El Yield de proceso aumenta claramente cuando se comienzan a utilizar revisiones de diseño y de código.

El COQ tiene tres componentes principales[1]:

1. *Costo de las Fallas (CF)*. El costo total de las fases de compilación y de pruebas;
2. *Costos de Evaluación (CE)*. El tiempo utilizado en revisiones de código y de diseño;
3. *Costos de Prevención*. El costo de identificar causas de defectos y acciones para prevenirlos en el futuro.

En la tabla 2.4 se muestran las métricas que se pueden obtener a partir del análisis del COQ[1].

Fórmulas COQ		
$CF = 100(\frac{TC+TP}{TTD})$	CF	Costo de Fallas
	TC	Tiempo de Compilación
	TP	Tiempo de Pruebas
	TTD	Tiempo Total de Desarrollo
$CE = 100(\frac{TRD+TRC}{TTD})$	CE	Costo de Evaluación
	TRD	Tiempo de Revisión de Diseño
	TRC	Tiempo de Revisión de Código
	TTD	Tiempo Total de Desarrollo
$TCOQ = CF + CE$	CE	Costo de Evaluación
	CF	Costo de Fallas
$CE \% = 100(\frac{CE}{TCOQ})$	CE	Costo de Evaluación como % del TTD
	TCOQ	Costo Total de la Calidad
$RF = \frac{CE}{CF}$	RF	Razón de Falla
	CE	Costo de Evaluación
	CF	Costo de Fallas

Tabla 2.4: Métricas del COQ

El Yield y el COQ son útiles, pero miden el trabajo que hiciste, no lo que estás haciendo. La Tasa de Revisión y el Radio de Calidad por Fase proveen una manera de dar seguimiento y control a los tiempos de revisión.

La tasa de revisión se utiliza principalmente para revisiones de código e inspecciones, y mide el número de Líneas de Código (LOC por sus siglas en Inglés) o páginas revisadas por hora.

El Radio de Calidad por Fase, Es el radio de tiempo utilizado en dos fases de proceso. El radio para la revisión es el tiempo utilizado para realizar revisiones dividido entre el tiempo de desarrollo.

El Índice de Calidad de Proceso (PQI por sus siglas en Inglés), es una métrica compuesta por cinco sub-métricas, la cual nos permite analizar el desempeño general de los procesos de una organización de software. El PQI se obtiene de multiplicar las cinco sub-métricas que se presentan en la tabla 2.5. El valor ideal es que el resultado sea 1.0; sin embargo, estudios realizados por Humphrey[1] indican que un valor mayor a 0.4 provee un valor adecuado de calidad.

Fórmulas PQI		
$DP = \frac{TD}{TP}$	DP	PQI de Diseño y Programación
	TD	Tiempo de Diseño
	TP	Tiempo Programación
$RD = 2(\frac{TRD}{TD})$	RD	PQI de Revisión de Diseño
	TRD	Tiempo de Revisión de Diseño
	TD	Tiempo de Diseño
$RP = 2(\frac{TRP}{TP})$	RP	PQI de Revisión de Programación
	TRP	Tiempo de Revisión de Programación
	TP	Tiempo de Programación
$DC = 20(10 + \frac{DC}{KLOC})$	DC	PQI para Defectos de Compilación por KLOC
	DC	Defectos de Compilación
	KLOC	Mil Líneas de Código
$DT = \frac{10}{5 + \frac{DT}{KLOC}}$	DT	PQI para Defectos de Pruebas por KLOC
	DT	Defectos de Pruebas
	KLOC	Mil Líneas de Código
$TPQI = DP \cdot RD \cdot RP \cdot DC \cdot DT$	TPQI	PQI Total

Tabla 2.5: Métricas del PQI

2.4.5 Prácticas para la Mejora de Calidad del PSP

Existen seis principios que recomienda el PSP para la Mejora de la Calidad Personal[1]:

1. Para tener calendarios predecibles, se tiene que planear y dar seguimiento al trabajo personal;
2. Para hacer planes precisos y que se les pueda dar seguimiento, estos planes tienen que ser detallados;
3. Para hacer planes detallados, se deben basar en datos históricos;
4. Para hacer un trabajo de alta calidad, se debe usar un proceso personal definido y medible;
5. Ya que el trabajo de mala calidad no es predecible, la calidad es un pre requisito para la predictibilidad.

Estos principios se promueven en el BM porque provee la infraestructura para dar seguimiento al trabajo personal, almacena datos históricos y permite el establecimiento de ciclos de vida.

2.4.6 Prevención de Defectos

Para prevenir la inyección de defectos se debe de revisar los datos de los defectos que más se encuentran durante el desarrollo y las pruebas. Esto nos ayuda a conocer cuales son los defectos más comunes y establecer estrategias como enfocarse en los errores que:

- Se encuentran en el programa final o en el periodo de pruebas;
- Aquellos que ocurren más frecuentemente;
- Aquellos que son más difíciles o costosos de corregir;
- Aquellos en los que se pueden realizar acciones preventivas sencillas;
- Aquellos que más nos molestan.

El BM está basado en esta filosofía. Fomenta las ideas de realizar revisiones personales a los programas. La revisión de defectos se hace con el fin de detectar los defectos, sin embargo no es suficiente, el BM también recomienda su registro, seguimiento y administración. Al registrar los defectos podemos generar estadísticas y reportes que permitan conocer los tipos de defectos mencionados anteriormente.

2.4.7 Técnicas de Detección de Defectos

Las técnicas de detección de defectos pueden ser definidas como la evaluación y la revisión de un producto de trabajo por parte de uno o más compañeros de trabajo calificados para realizar la actividad[17].

Existen diferentes tipos de técnicas de detección de defectos, unas más formales y rígidas que otras. Evidentemente la diferencia radica en la forma de conducir la revisión del producto de trabajo. También existen diferentes clasificaciones acerca de las técnicas. Una posible clasificación consiste en separar las técnicas que involucran a una sola persona y las técnicas en las que participan dos o más miembros del equipo. Los tipos más utilizados son: La revisión personal y la revisión entre colegas.

La revisión personal consiste en examinar el producto de trabajo antes de entregarlo a cualquier otro miembro del equipo, ya sea para su lectura, compilación, revisión, implementación o prueba.

Existen varias técnicas de detección de defectos que se realizan entre colegas y/o miembros de un equipo de desarrollo[18]. Estas técnicas se puede clasificar de acuerdo a su formalidad, de acuerdo [18] con existen 6 diferentes tipos de técnicas:

- Revisión ad hoc;
- Revisión general;
- Revisión de parejas;
- Revisión de equipo;
- La Caminata y la inspección son explicadas más detalladamente a continuación.

La caminata es un tipo de revisión en el que precisamente una persona (preparada especialmente para ello) pasa por cada una de las partes del producto, exponiéndolo a una audiencia. Mediante esta técnica se pueden obviar muchos detalles, con lo cual se reduce el tiempo de revisión. Sin embargo, esto puede ser contraproducente si el objetivo es precisamente detectar defectos que residen en los detalles. Al mismo tiempo, el proceso de revisión está definido por el producto de trabajo siendo revisado, a diferencia de la inspección, en el que el proceso se determina por los puntos a revisar[19].

La Inspección es probablemente la técnica más formal de detección de defectos de software[18]. Consiste de los siguientes pasos[17]:

1. *Planeación*: Seleccionar dónde, cuándo y quiénes participarán;
2. *Resumen*: Revisión general para que los revisores se familiaricen con el producto de trabajo;
3. *Preparación*: Cada revisor lee el producto de trabajo e identifica posibles defectos. Todo esto generalmente mediante listas de chequeo. La persona que realiza el papel de líder, se prepara para la junta de revisión;

4. *Junta*: El líder modera la junta y realiza la revisión del producto. Los revisores lo interrumpen para discutir los defectos encontrados. Lo más importante es que NO se permite discutir posibles soluciones para ningún defecto;
5. *Re-trabajo*: El autor del producto realiza las correcciones pertinentes;
6. *Seguimiento*: El autor del producto notifica al líder de las correcciones y éstas son revisadas.

Tanto la caminata como la inspección aumentan su efectividad cuando se utilizan listas de chequeo. El BM permitirá la creación de plantillas que sirvan de guía para la realización de estas técnicas. La funcionalidad será explicada más a detalle en la sección [REFERENCIA].

2.4.8 ¿Por qué Revisar los Programas?

Los programas se revisan porque es la manera más rápida y más barata para encontrar y corregir problemas antes de diseñar una función equivocada o implementar un diseño incorrecto. Un poco de tiempo invertido revisando un programa puede ahorrar mucho tiempo durante las fases de compilación y pruebas. Aún más importante, cuando todos en el equipo de desarrollo hacen revisiones de diseño y de código a conciencia, el tiempo de integración y de pruebas de sistema es reducido en un factor de cinco a diez.

2.4.9 Principios de la Revisión

Los principios de las revisiones personales son los siguientes[1]:

- Revisar personalmente todo el trabajo propio antes de pasar a la siguiente fase de desarrollo;
- Intentar lo mejor posible corregir todos los defectos antes de dar el producto de trabajo a otra persona en el equipo de desarrollo;
- Utilizar una lista de chequeo personal y seguir un proceso estructurado de revisión;

- Seguir las buenas prácticas de la revisión: revisar en incrementos pequeños, hacer las revisiones en papel y hacerlas cuando estás descansado;
- Medir el tiempo de la revisión, el tamaño de los productos revisados y el número y tipo de defectos encontrados y perdidos;
- Usar los datos de las mediciones para mejorar el proceso personal de revisión;
- Diseñar e implementar los productos para que sean fáciles de revisar;
- Revisar los datos para identificar las formas de prevenir defectos.

Esta es la herramienta principal que propone el BM para la detección de defectos. El BM tiene una plantilla por default para realizar revisiones personales en lenguajes de programación de propósito general. Esto se explica más a detalle en la sección [REFERENCIA].

2.4.10 Lista de Chequeo de Revisión de Código

Una lista de chequeo es una forma especializada que se usa para hacer las revisiones. La lista de chequeo también ayuda a disciplinar el trabajo y guiar en el proceso de revisión.

Para la revisión personal es necesario que cada persona desarrolle su propia lista de chequeo, para que cada persona se enfoque en los errores que más comete, o en los que quiere evitar según la estrategia que se siga, y que se completen todos los elementos de la lista. Para hacer un mejor uso de la lista de chequeo se tiene que dividir la lista en secciones con características similares para concentrarte en los mismos tipos de errores en cada pasada al documento a revisar (ya sea código, diseño o requerimientos). Para elaborar una lista de chequeo se puede tomar como base los tipos de defectos mostrados en la tabla 2.6 [1].

La lista de chequeo es construida a partir de esta lista de defectos. Se deben proponer distintas actividades para enfocarse a remover los distintos tipos de defectos. La tabla 2.7 presenta un ejemplo de lista de chequeo para lenguajes de propósito general propuesta por Humphrey[1].

Número de Tipo	Nombre del Tipo	Descripción
10	Documentación	Comentarios y mensajes.
20	Sintaxis	Ortografía, puntuación y tipos.
30	Paquete	Administración, librerías y versiones.
40	Asignación	Declaración, nombres duplicados y límites.
50	Interface	Procedimientos, referencias, I/O y formatos.
60	Chequeo	Mensajes de error y chequeos inadecuados.
70	Datos	Estructura y contenido.
80	Función	Lógica, apuntadores, ciclos, etc.
90	Sistema	Configuración, tiempo y memoria.
100	Medio Ambiente	Diseño, compilación y pruebas.

Tabla 2.6: Clasificación de Defectos del PSP

Lista de Chequeo para la Revisión de Código	
Nombre del Revisor	
Nombre del Programa	
Lenguaje	
Propósito	Ser una guía para la revisión de código
General	<ul style="list-style-type: none"> - Revisa el programa por cada categoría. - No intentar revisar dos categorías a la vez. - Cada que se complete un paso palomear el cuadro.
Documentación	<ul style="list-style-type: none"> - Los métodos están documentados correctamente.
Sintaxis	<ul style="list-style-type: none"> - Todas las líneas del programa terminan en ";". - Todas las llaves, corchetes y paréntesis tienen su pareja. - El código está correctamente indentado.
Paquete	<ul style="list-style-type: none"> - El programa se encuentra en la carpeta correcta. - Todas las librerías requeridas están presentes.
Asignación	<ul style="list-style-type: none"> - Las variables tienen el tipo de dato correcto. - Las variables siguen las convenciones de nombrado. - Todas las variables están inicializadas. - Las variables tienen el alcance adecuado.
Interface	<ul style="list-style-type: none"> - Los archivos leídos o escritos existen. - Las llamadas a funciones son correctas.
	<ul style="list-style-type: none"> - Las llamadas a funciones tienen los parámetros adecuados.
Chequeo	<ul style="list-style-type: none"> - Los métodos utilizados manejan excepciones. - Los mensajes de error mandados son útiles.
Datos	<ul style="list-style-type: none"> - La base de datos utilizada existe. - Las consultas a la base de datos son correctas.
Función	<ul style="list-style-type: none"> - La lógica de los métodos es correcta. - Los apuntadores están definidos correctamente. - Los ciclos respetan el tamaño de los arreglos.
Sistema	<ul style="list-style-type: none"> - El programa utiliza correctamente la memoria disponible.

Tabla 2.7: Lista de Chequeo para la Revisión de Código de PSP

La lista de chequeo aparte de ser personalizada, tiene que ser actualizada constantemente. La persona al seguir los procesos de calidad tiene una mejoría notable en la forma de hacer su trabajo, por lo tanto los errores que se comenten en el tiempo van cambiando y la lista tiene que ser actualizada. También se tiene que tener en cuenta que las listas de chequeo cambian notablemente de un proyecto a otro, dependiendo del lenguaje de programación que se utilice entre otros factores.

La intención de la revisión de código es asegurar que todos los detalles son correctos. Una vez que se definen las prácticas, deben de ser incorporadas en los estándares y ser checados en las revisiones.

La estrategia de la revisión es:

- Para asegurar que el código cubre todo el diseño, se revisa cada método para asegurar que todas las funciones requeridas están incluidas;
- Para checar las librerías a incluir, examinar cada método para asegurar que existan las inclusiones necesarias para cada función de la librería;
- Para checar problemas de inicialización, realizar una caminata por la lógica de todos los métodos.

Esta estrategia debe de ser utilizada en las revisiones registradas en el BM.

2.4.11 Evaluando las Revisiones Personales

Algunas medidas útiles para evaluar las revisiones son:

- El tamaño del programa revisado;
- El tiempo de revisión en minutos;
- Número de defectos encontrados;
- Número de defectos en el programa que se encontraron después, en otras palabras aquellos que no fueron encontrados en la revisión.

Las métricas formales que se proponen por PSP son: Yield de Revisión, Densidad de Defectos, Tasa de Defectos y Tasa de Revisión.

Yield de revisión es el porcentaje de defectos en el producto que fueron encontrados en la revisión. Un Yield alto es bueno, uno malo es pobre. La meta del Yield debe de ser el 100 %. La meta de PSP es encontrar y corregir todos los errores antes de la primera compilación o las pruebas; es decir, un yield de proceso ed 100 %. Para hacer un estimado de Yield para cualquier fase, debes de asumir el Yield de la fase anterior. Inicialmente es común un Yield de fase de 50 % y conforme se va madurando la técnica es posible tener 70 % o más[1].

La Tasa de Defectos es el número de defectos que se encontraron en el código por LOC.

La Tasa de Revisión es el número de líneas que se revisan por hora. El ideal es entre 200 y 400 líneas por hora.

La efectividad de los métodos de revisión es el radio de defectos removidos por hora en la revisión.

2.4.12 Efectividad de la Revisión

Las revisiones de códigos son inherentemente mejores que las pruebas[1]. Depuración es el proceso de encontrar código defectuoso que causa que el programa se comporte impropriamente. La cantidad de tiempo que se gasta realizando el debugging generalmente tiene poca relación con la complejidad del defecto[1].

La meta de la revisión es remover el máximo número de defectos, en otras palabras llegar a pruebas con cero defectos. En una revisión, personalmente se revisa el programa que una ha generado. Una inspección es una revisión en equipo de un programa. Después de las revisiones personales, la inspección es la técnica más valiosa que un equipo de desarrollo de software puede utilizar.

2.4.13 Diseño de Software

La principal herramienta a disposición de los ingenieros de software son las abstracciones. Se pueden crear abstracciones arbitrariamente y combinarlas en abstracciones más grandes.

Si se cumplen con las capacidades de los sistemas que se soportan entonces también se pueden crear las estructuras lógicas que necesarias.

El principal problema reside en la escala o el tamaño de los sistemas de software. La forma de subdividir los sistemas tiene que ser coherente y realmente ayudar a resolver los problemas de complejidad. Ya que si en el caso de un sistema de 1,000,000 LOC, creas 500 tipos de 10 LOC, entonces tendrías que diseñar un sistema de 100,000 LOC, lo que parece una ayuda, pero en realidad se fuerza a los diseñadores a relacionar coherentemente los 500 tipos distintos. Para que la escalabilidad sea útil, no solamente se debe de capturar los requerimientos físicos de la escalabilidad, si no capturar un nivel de funcionalidad significativa en las partes.

La forma más clásica es utilizar la regla de divide-y-vencerás. En otras palabras, el sistema se tiene que dividir en partes más pequeñas, las cuales deben de ser vistas como subsistemas coherentes o componentes. Si esto se hace correctamente, se reduce la complejidad en un factor de diez o más. Sin embargo hacer esto correctamente es una tarea difícil, a la que se llama diseño. El diseño de calidad tiene dos partes: la calidad del concepto de diseño y la calidad de la representación del diseño.

Producir un diseño claro, completo y libre de defectos es un paso necesario para construir productos de calidad. Crear diseños detallados hace más lento el proceso individual de desarrollo, sin embargo, si todo el equipo de desarrollo hace lo mismo, acelerará el trabajo en equipo ya que facilitará el trabajo de integración y pruebas de sistema.

2.4.14 El Proceso de Diseño

El diseño de software es una actividad creativa la cual no puede ser reducida a un proceso rutinario. Sin embargo, esta actividad, no tiene que carecer por completo de estructura. El diseño generalmente inicia revisando el propósito del producto, obteniendo datos relevantes, produciendo una vista general del diseño y llenando los detalles. Para diseños complejos, los buenos diseñadores siguen un proceso dinámico. Ellos trabajan en un nivel conceptual por un periodo de tiempo y después ahondan en cada parte.

Ocasionalmente, el diseñador no va a ser capaz de especificar una función hasta que se haya diseñado, construido y probado. En estos casos se deben de construir prototipos para

estas funciones. Antes de desarrollar un prototipo, se debe de especificar su propósito y las preguntas que va a responder.

2.4.15 Niveles de Diseño

Los diferentes niveles de diseño son[1]:

1. *El diseño conceptual*. Es un concepto de planeación que se utiliza para estimar el tamaño del producto.
2. *Diseño de alto nivel (HLD)*. Este diseño tiene como objetivo dividir el sistema que se está desarrollando en partes más pequeñas o componentes.
3. *Diseño detallado (DLD)*. Este nivel de diseño toma los componentes especificados por el HLD y define como construirlos.

Los ciclos de vida por default propuestos en el BM contienen etapas de diseño previas a la programación y es sugerido que esto se haga para asegurar una alta calidad en el software elaborado. Esto será descrito más a detalle en la sección 3.3.4.

2.4.16 Calidad y el Diseño

Un diseño con un nivel de detalle adecuado es una excelente guía durante la construcción del sistema de software, ayudando a los programadores a estar seguros de cuando terminaron el programa y si tiene todas las funcionalidades requeridas.

Cuando no se realiza un diseño previo a la programación, se comienza a diseñar al momento de programar y no se da la suficiente atención a los posibles detalles, casos extremos y otras eventualidades que pueden ocasionar que un sistema falle.

Es por esto que una herramienta enfocada a mejorar la calidad del software como el BM haga un énfasis especial en diseñar antes de programar y que este diseño sea un diseño de calidad.

Los factores que afectan la calidad en el diseño son la precisión y la completez[1]. El diseño de software debe de contener una solución completa y precisa al problema.

La falta de un diseño preciso es la fuente de muchos errores de implementación. Para lograr una representación adecuada del sistema requieren 3 cosas: Producir un diseño coherente, registrar toda la información de diseño y registrar esa información en una forma precisa y entendible.

Una vez generado el diseño se debe analizar su corrección, completez y consistencia, para asegurarse que la notación lógica sea precisa. El diseño de alta calidad debe tener mínima redundancia.

2.4.17 Verificación de Diseño

Para programas largos y complejos, hasta los métodos de verificación de diseño que más consumen tiempo y más trabajo requieren son más efectivos y rápidos que las pruebas. Los siguientes métodos de verificación son relativamente rápidos de aprender:

- Estándares de diseño;
- Verificación por tabla de ejecución;
- Verificación por tabla de seguimiento;
- Verificación de máquinas de estado;
- Verificación analítica.

Los programas tienen que ser verificados porque los desarrolladores tienen la creencia que pocos defectos en un programa son aceptables mientras el programa se ejecute normalmente bien[1]. Pero qué pasa cuando 1 o 2 defectos por cada 100 líneas de código se convierten en 10,000 a 20,000 en 1,000,000 de líneas de código.

Es recomendable utilizar técnicas de verificación de diseño para reducir el número de defectos y aumentar la calidad del programa. El problema no es que las pruebas sean malas, sino que no son suficientes. Imagina analizar todas las posibilidades de ejecución de cierto programa, eso nos llevaría demasiados años. Por eso es importante utilizar técnicas de verificación para reducir el número de defectos, que además se traduce en la reducción

del costo para encontrar esos defectos, ya que se encuentran en una etapa temprana de desarrollo.

El diseño de un programa contra las bases de los estándares de cada organización. Todos los estándares definen convenciones para el producto, estándares de diseño del producto y estándares para reutilización de código.

Las convenciones para el producto final incluyen las interfaces de usuario, la manera de atacar errores, convenciones de nombres, procedimientos de instalación y la ayuda. El estándar de diseño del producto abarca desde las convenciones hasta la arquitectura del sistema. Los estándares para reutilización de código precisan que las partes a reutilizar estén completamente definidas, sean de la mayor calidad y estén soportadas adecuadamente.

Una técnicas de verificación es la Verificación Analítica[1] o verificación de ciclos. La verificación de ciclos nos ayuda a asegurarnos que el programa no caerá en ningún ciclo infinito, no se quedará atorado y no presentará ningún comportamiento extraño. En general, la verificación de ciclos consiste en verificar que las precondiciones del ciclo se cumplan siempre, que se garantice la terminación del ciclo para cualquier argumento que afecte al ciclo y que se mantenga la identidad cuando se cumple la condición de terminación, es decir, que no se ejecute una o más veces de las correctas.

La figura 2.1 contiene un fragmento del código del BM. A partir de este código se realizará una Verificación Analítica de este.

```
1  if (projectOrder >= 0 && phase.getProjectOrder() != projectOrder){
2
3      int actualOrder = phase.getProjectOrder();
4      Project project = projectDAO.findById(phase.getProject().getProjectId());
5      ;
6      List<Phase> phases = project.getPhases();
7
8      if (projectOrder > actualOrder){
9          for (Phase p: phases){
10             if (p.getProjectOrder() > actualOrder && p.getProjectOrder() <=
11                 projectOrder){
12                 p.setProjectOrder(p.getProjectOrder() - 1);
13             }
14         }
15     }
```

```
12     }
13 }
14 else if (projectOrder < actualOrder) {
15     for (Phase p : phases) {
16         if (p.getProjectOrder() < actualOrder && p.getProjectOrder() >=
17             projectOrder) {
18             p.setProjectOrder(p.getProjectOrder() + 1);
19         }
20     }
21
22     phase.setProjectOrder(projectOrder);
23 }
```

Listing 2.1: Ejemplo Verificación Analítica

El código presentado realiza el ajuste del orden de las fases dentro del proyecto. Por ejemplo, si el proyecto tiene tres fases: A, B y C; donde A es la primer fase, B es la segunda fase y C la tercera; entonces si agregamos una fase X entre B y C, el código debe de modificar el campo de las fases para que especifique que A es la primer fase, B la segunda, X la tercera y C la cuarta. El código realiza lo siguiente:

- De la línea 3 a la 5 se obtiene el orden actual de la fase (la variable `actualOrder`), el objeto del proyecto y la lista de fases;
- En la línea 7 se compara el orden actual de la fase con el orden nuevo que esta tendrá (la variable `projectOrder`). Si `projectOrder` es mayor a `actualOrder` se ingresa al ciclo de la línea 8;
- El ciclo de la línea 8 recorre la lista de fases y si el orden actual de la fase (la llamada `p.getProjectOrder()`) es mayor a `actualOrder` y menor o igual a `projectOrder` entonces el orden de la fase tiene que ser restado en uno.;
- El ciclo de la línea 15 recorre la lista de fases y si `p.getProjectOrder()` es menor a `actualOrder` y mayor o igual a `projectOrder` entonces el orden de la fase tiene que ser aumentado en uno.

- En la línea 14 se compara `actualOrder` con `projectOrder`. Si `projectOrder` es menor a `actualOrder` se ingresa al ciclo de la línea 15;

Ahora se realizará la Verificación Analítica del código:

1. *Precondiciones del ciclo.* La precondición es que exista una lista con fases, esta lista siempre existirá ya que la misma fase de donde se obtienen las distintas variables es parte de esta lista.
2. *Terminación del ciclo.* El ciclo siempre tendrá a un final ya que este simplemente recorre la lista de fases recibida.
3. *Identidad del ciclo.* El ciclo se ejecuta una sola vez por fase por lo que su identidad se mantiene.

Cuando se diseñan programas complejos, siempre se cometen errores. Por lo tanto, siempre será importante realizar revisiones completas a todos los diseños que se hagan. Además, es importante contar con una estrategia de verificación que permita maximizar los resultados de la verificación. Esta estrategia puede no ser la misma para diferentes tipos de programas.

Sin embargo, el mejor momento para realizar la verificación y los análisis siempre será mientras se produce el diseño. Mantener el registro de resultados actualizado y seguir procedimientos para la revisión del diseño serán dos tareas que maximizarán los resultados de la verificación.

2.5 Defectos de Software

El BM como su nombre lo dice, hace un gran énfasis en la administración adecuada de los defectos encontrados durante el desarrollo de software. A continuación se presentarán los conceptos principales utilizados en la herramienta para hacer una administración adecuada de defectos.

2.5.1 Clasificación Ortogonal de Defectos

Normalmente, los defectos representan el aspecto no deseado en el manejo de calidad del software. Una herramienta que nos permite atacar las debilidades de los modelos antes mencionados es la Clasificación Ortogonal de Defectos, la cual está diseñada para capturar todos los atributos de los defectos y permitirnos hacer análisis matemáticos sobre ellos[20].

La clasificación ortogonal de defectos nos permite clasificar 8 tipos de atributos diferentes, que están divididos en 2 clases: La sección de apertura y la sección de clausura[20].

En la sección de apertura se encuentran los atributos que son registrados cuando se encuentra el defecto, y antes de resolverlo. Los atributos son[20]:

- *Actividad.* Se refiere a la actividad que se estaba realizando cuando se encontró el defecto. Un ejemplo de estas actividades sería inspecciones de código, pruebas unitarias, pruebas de sistema, etc.
- *Trigger.* Se refiere al ambiente o condición existente que ocasiona que el defecto aparezca, aquella condición necesaria para reproducir el defecto. El método ya tiene algunos activadores definidos para cierto tipo de actividades, aunque también se pueden usar los propios.
- *Impacto.* Aquí se selecciona el impacto que el defecto podría haber tenido en el cliente final si no se arreglara. En caso de que el defecto lo haya detectado un cliente, se pone el impacto actual que el defecto tuvo en el cliente.

En la sección de clausura se encuentran los atributos que son registrados después de que el defecto ha sido arreglado. Los atributos son[20]:

- *Target.* Representa la entidad de alto nivel que se arregló. Por ejemplo, el diseño o el código del programa.
- *Tipo de defecto.* Representa la naturaleza de la corrección que se realizó. Para este campo existen valores predeterminados, por ejemplo: asignación o inicialización, que se refiere a que el valor de una variable fue corregido; algoritmo/método, significa

que se implementó o corrigió un algoritmo; tiempo/serialización, que nos indica que se tuvo que implementar métodos de serialización en un recurso compartido.

- *Qualifier*. Aplica al tipo de defecto, y captura el calificativo que describe al defecto. Para este atributo existen 3 valores posibles:

- *Omisión*. Indica que el defecto se debió a una omisión.
- *Incorrecto*. Indica que valores incorrectos se usaron.
- *Extraño*. Indica que el defecto se debe a algo no relevante al código.

- *Fuente*. Identifica el origen del target que tenía el defecto. Para este atributo también existen ciertos valores predeterminados:

- *Desarrollado internamente*. Nos dice que el defecto fue introducido por el equipo de desarrollo.
- *Reusado de una librería*. Nos indica que el defecto pertenecía a una librería de reuso.
- *Outsourced*. Indica que el defecto fue introducido por una empresa que hizo servicios de outsourcing.
- *Ported*. Indica que el defecto tiene que ver con una parte que fue validada en un ambiente diferente.

- *Edad*. Identifica el historial del target que tenía el defecto. Para este campo existen los siguientes valores:

- *Base*. Indica que el defecto está en un aparte del producto que no ha sido modificada en el proyecto actual, y no es parte de una librería de reuso. Es un defecto latente.
- *Nuevo*. El defecto fue introducido en el proyecto actual.
- *Re-escrito*. El defecto fue introducido por re-diseñar o re-escribir una función con el objetivo de mejorar su diseño o calidad.

- *Re-arreglado*. El defecto fue introducido al proveer una solución a un defecto previo.

El BM extiende la clasificación ortogonal de defectos para dar seguimiento a los que sean registrados en el sistema. El seguimiento que se le dará a los defectos dentro del BM será revisado con más detalle en la sección 3.3.6.

2.5.2 Inyección de Defectos y Eficiencia en la Remoción de Defectos

La frase inyección de defectos se refiere al número probable de defectos que serán encontrados durante el desarrollo de aplicaciones de software[21]. La tabla 2.8 nos muestra el número promedio de defectos inyectados durante el desarrollo por puntos de función[22].

Requerimientos	1.00
Diseño	1.25
Codificación	1.75
Documentación	0.60
Correcciones Erróneas	0.40
Total	5.00

Tabla 2.8: Potencial de Inyección Defectos Promedio

La frase eficiencia en la remoción de defectos se refiere al porcentaje de defectos potenciales que serán removidos antes de que la aplicación de software entre en producción[21]. Algunos defectos son más difíciles de remover que otros, como se muestra en la tabla 2.9, donde se compara la eficiencia de remoción de defectos contra diferentes categorías de defectos[22].

Origen del Defecto	Potencial	Eficiencia en la Remoción	Defectos Restantes
Requerimientos	1.00	77 %	0.23
Diseño	1.25	85 %	0.19
Codificación	1.75	95 %	0.09
Documentación	0.60	80 %	0.12
Correcciones Erróneas	0.40	70 %	0.12

Tabla 2.9: Efectividad en la Remoción de Defectos

2.5.3 El Costo Real de los Defectos de Software

Es obvio que mientras una aplicación defectuosa crece y evoluciona es más costoso repararla [21]. Un defecto que cueste \$1 repararlo en la computadora del programador costará \$100 repararlo ya que se haya incorporado a la aplicación completa, y costará algunos miles de dólares si llega a producción[9]. Este comportamiento se muestra en la figura 2.1 donde encontramos el costo de remover defectos de software según la fase de inyección y de remoción[9].

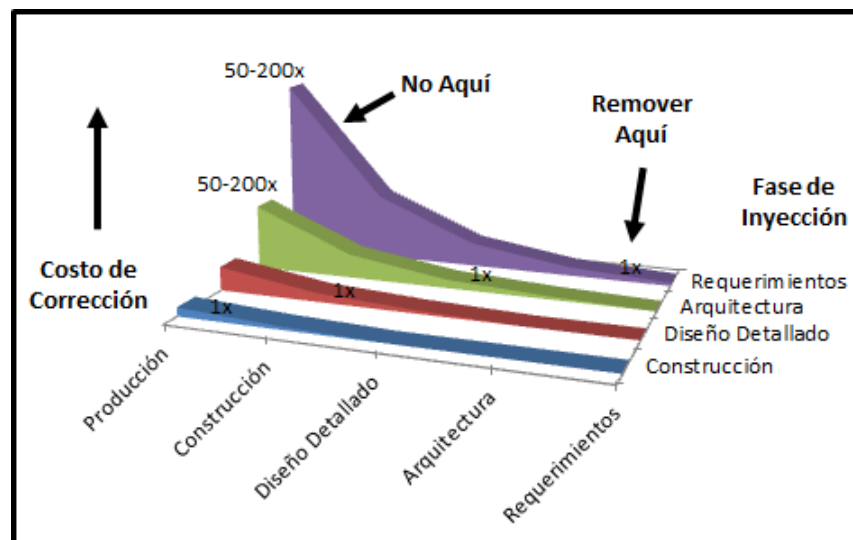


Figura 2.1: Costo de la Remoción de Defectos por Fase

El costo de remover un defecto de software crece exponencialmente cada fase que este se mantiene dentro del ciclo de desarrollo sin ser detectado[23]. En un proyecto típico de software, el 80 % del costo total es utilizado en la corrección de defectos[21].

Es por esto que el BM sugiere una cultura de revisión y prevención, la cual tiene como objetivo evitar la introducción de defectos, removerlos tan rápido como sea posible si es que estos son inyectados y así evitar el aumento exponencial de costo y esfuerzo. El BM sugiere que la calidad en el desarrollo de software es mucho más que simplemente introducir una fase de pruebas. Las iniciativas de calidad deben existir durante todo el proceso de desarrollo para alcanzar la meta de un software con cero defectos.

2.6 Costo de la Calidad de Software

El Costo de la Calidad (por sus siglas en Inglés CoQ) representa los costos en los que incide una organización al tener que repetir un proceso para realizar el trabajo correctamente[21]. El CoQ es un término que nos permite evaluar la economía involucrada para producir software de alta calidad[21]. El CoQ se divide en dos tipos principales: El costo de la conformidad y el costo de la no conformidad[24].

$$CoQ = Costo_{Conformidad} + Costo_{No-conformidad}$$

A su vez, el costo de la conformidad se divide en costos de prevención y costos de evaluación, mientras que el costo de la no conformidad se divide en fallas internas y externas[25]. Esto se muestra en la figura 2.2:

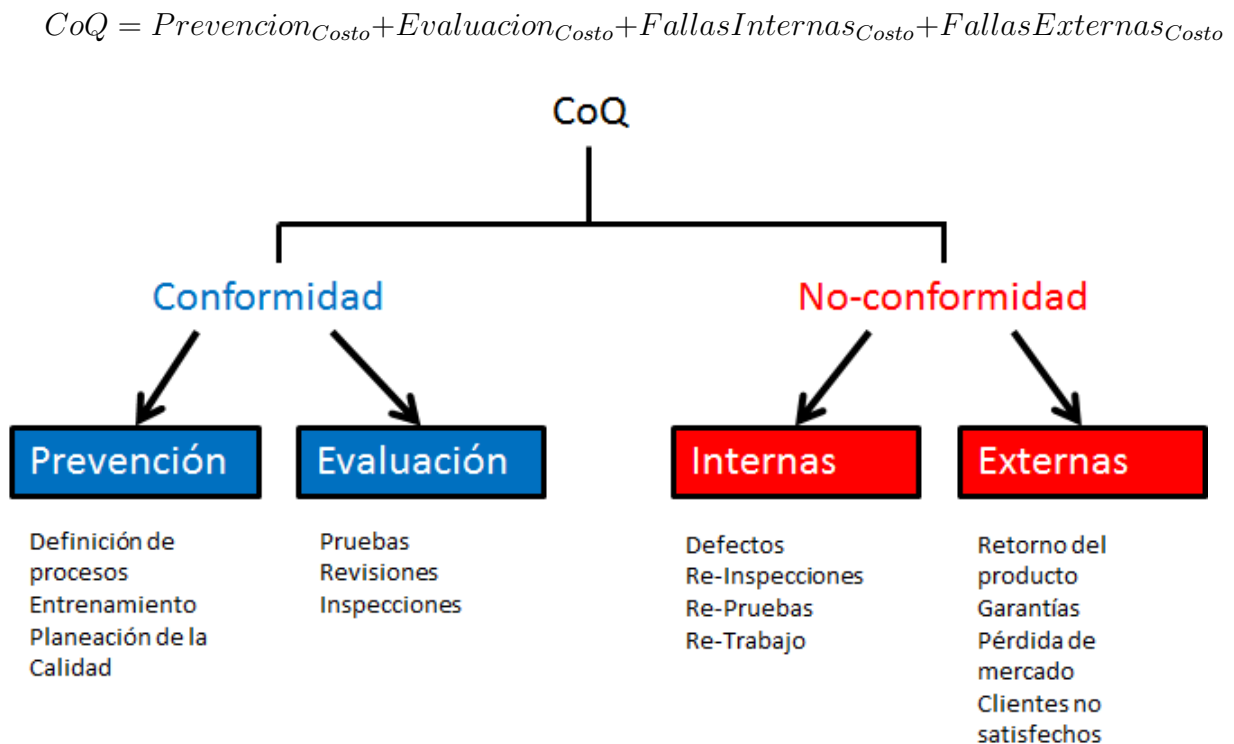


Figura 2.2: Costo de la Calidad de Software

Los significados los diferentes tipos de costos son los siguientes[26]:

- *Costos de prevención.* Son los costos asociados con la planeación de la calidad; el diseño, la implementación y la administración de la calidad del sistema; la auditoría del sistema; encuestas a los proveedores y las mejoras de proceso.

- *Costos de evaluación.* Son los costos asociados con la medición, evaluación y revisión de los productos para asegurar su conformidad con los estándares de calidad y requerimientos de desempeño.
- *Costos de falla.* Son las pérdidas asociadas con la creación de un producto que no cumpla con los criterios de conformidad. Se dividen en internos y externos.
- *Costos de falla internos.* Son los costos asociados con las fallas y defectos del proceso, equipo, producto y materiales que no cumplen con los estándares de calidad o los requerimientos.
- *Costos de falla externos.* Son generados por productos, servicios y procesos defectuosos cuando el usuario los utiliza. Incluyen garantías, quejas, devoluciones y reemplazos.

La figura 2.3 es una representación gráfica del CoQ propuesta por varios investigadores[27, 25, 24].

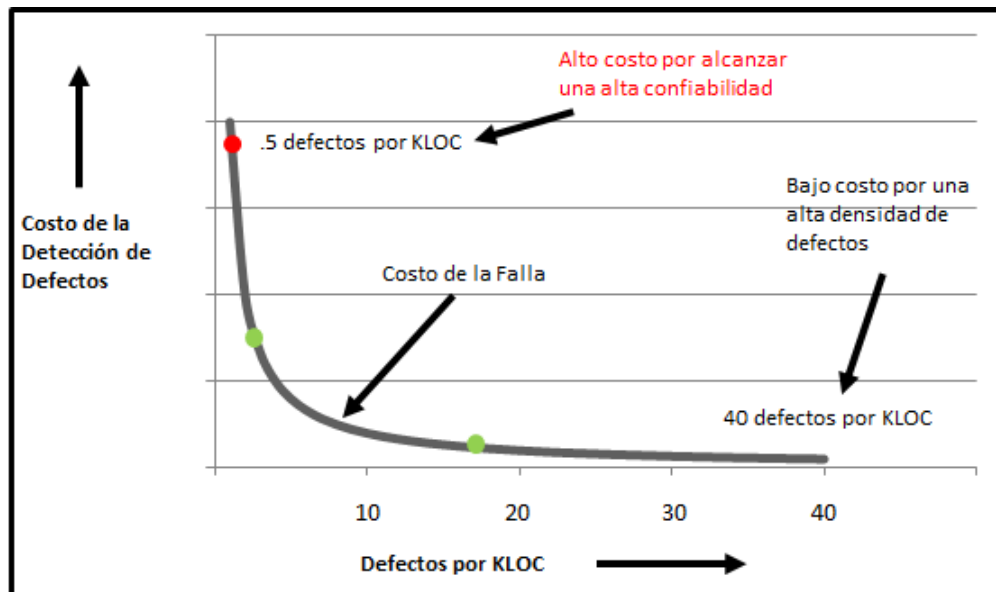


Figura 2.3: El costo de la alta confiabilidad

La figura anterior nos muestra que para alcanzar una alta confiabilidad y cero defectos (cerca del punto rojo) el costo es muy alto, pero alcanzar un nivel razonable de calidad (entre los dos puntos verdes) no requiere un costo muy alto.

2.6.1 De las Pruebas a la Prevención

Anteriores esquemas de calidad de software sugerían que una fuerte etapa de pruebas era la mejor acción posible para asegurar la calidad en el software, sin embargo, una encuesta dirigida por [21] a distintas organizaciones dedicadas al desarrollo de software encontró que la mayoría de estas organizaciones estaban de acuerdo que una cultura que favoreciera las pruebas antes que la prevención de los defectos provocaba un programa poco productivo de calidad.

Los costos para dar calidad y los costos causados por falta de la calidad tienen una relación inversa: mientras que la inversión en alcanzar calidad aumenta, los costos provocados por la falta de calidad disminuyen[21]. Este modelo teórico se muestra en la figura 2.4[21]. La figura muestra que mientras los costos de prevención y evaluación aumentan, los costos de las fallas disminuyen hasta que se alcanza un punto óptimo, después del punto óptimo aumentar la inversión en prevención y evaluación no tienen tan buenos resultados [21].

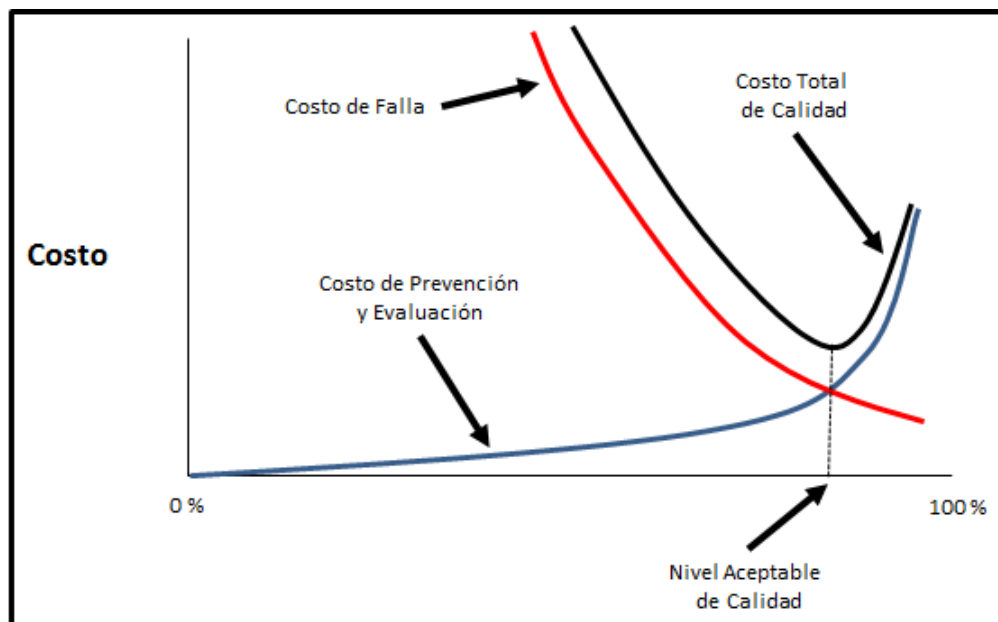


Figura 2.4: Modelo de la Calidad de Software

Cuando se comienza a invertir en evaluación provoca que las fallas internas aumenten ya que se detectan más defectos en etapas tempranas, pero remover defectos en etapas tempranas es mucho más barato que hacerlo en pruebas o mantenimiento[21]. En general

las actividades de evaluación disminuyen las fallas externas y el total de fallas disminuye. Una inversión pequeña en prevención y evaluación produce grandes disminuciones de costo total de calidad[21].

Fomentar una cultura donde la prevención de los errores sea más importante que el proceso de pruebas es uno de los objetivos de este trabajo de Tesis.

2.6.2 Análisis del Costo de la Calidad

El análisis del costo de la calidad es el concepto de estudiar los costos relacionados con la calidad como medios de comunicación entre el departamento de calidad y la gerencia de las organizaciones[21]. El objetivo de hacer un análisis del costo de la calidad no es reducir el costo, si no asegurarnos de que se invierta en el tipo correcto de costo y se maximice el beneficio obtenido de esa inversión. La mejor estrategia para esto es cambiar las actividades relacionadas con las fallas por actividades de prevención y evaluación.

Un análisis costo beneficio se realiza para determinar que tan bien, o que tan mal, una acción resultará. Un análisis de costo beneficio encuentra, cuantifica y agrega los factores positivos. Estos son los beneficios, entonces cuantifica y substraer todos los negativos, en otras palabras los costos. La diferencia entre estos indica si la acción es recomendable o no[21].

Una consideración clave en el análisis del costo de la calidad es la visibilidad. La visibilidad obtenida gracias al análisis del costo de la calidad permite al equipo de aseguramiento de la calidad enfocar su atención en aquellas actividades que descubren y corrigen la causa raíz de los defectos. La causa raíz puede ser utilizada para determinar como el proceso de desarrollo puede ser mejorado para prevenir nuevos defectos[21].

Knox propone un modelo teórico en el cual demuestra la inversión en las distintas actividades del CoQ según el nivel de CMMI[27]. Este se muestra en la figura 2.5 donde podemos observar como se reduce el CoQ total al ir aumentando las actividades de evaluación y prevención según el nivel de CMMI.

Para hacer una mejora real en la calidad de software debemos enfocarnos en dos mejoras de proceso[21]: Prevención de defectos y remoción de defectos.

La prevención de defectos se refiere a las tecnologías y metodologías que reducen el

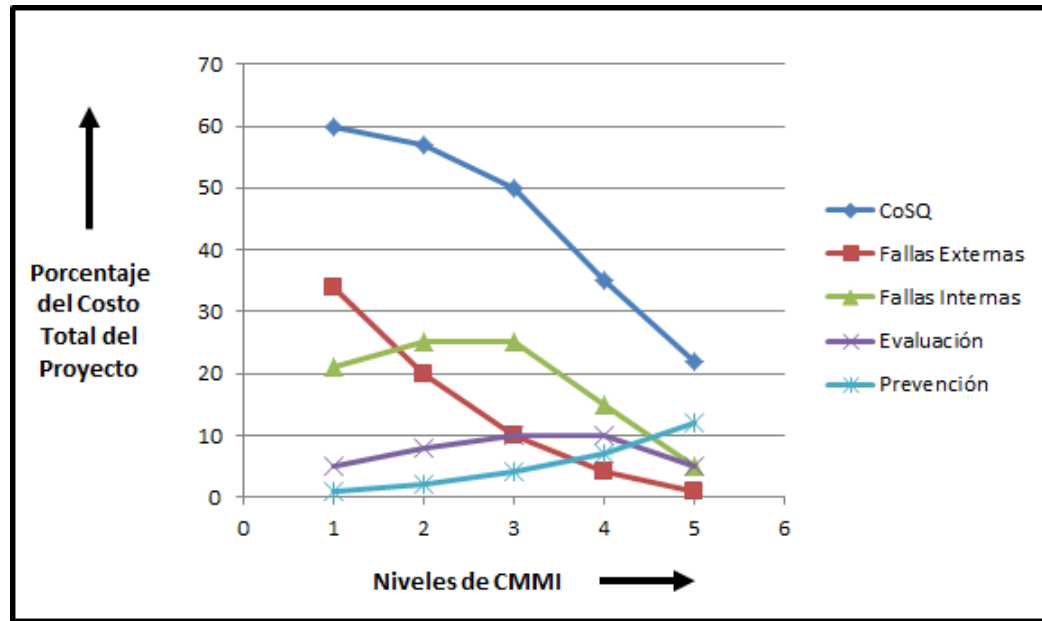


Figura 2.5: CoQ por nivel de CMM

número de defectos que deben de ser eliminados. Ejemplos de estos métodos pueden ser métodos formales de diseño, generación automática de código a partir de especificaciones formales, la capacitación y la creación de una conciencia sobre los defectos inyectados.

La remoción de defectos se refiere a los métodos que pueden aumentar los niveles de eficiencia en la remoción de defectos al introducir diferentes tipos de revisiones.

2.6.3 Análisis del Retorno de Inversión

Hay dos beneficios principales de tener una alta calidad en el software: Tiempo y dinero. El retorno de inversión (por sus siglas en Inglés ROI) analiza los ahorros en costo y en calendario que se tienen al implementar ciertas actividades de calidad[21]. El ROI se expresa en términos de esfuerzo ya que es el mayor costo de un proyecto de software.

Una forma de calcular el ROI de un proyecto es la siguiente[21]:

$$ROI = \frac{OriginalTotalCoQ - NewTotalCoQ}{OriginalTotalCoQ}$$

Este modelo nos muestra el porcentaje de ahorro del costo de calidad total del proyecto. Para ilustrarlo se muestra un ejemplo en la tabla 2.10[21]. En esta tabla se muestran dos casos hipotéticos, en ambos se trata de un proyecto con mil defectos. El primero muestra el CoQ de una empresa que no invierte en prevención y evaluación, mientras que en el

segundo caso se muestra el escenario donde se realiza la inversión. De esta manera se calcula el ROI, realizando la comparación de los costos que se hubieran tenido sin una estrategia de calidad, contra el CoQ real.

Análisis del ROI		
Recursos	Caso1	Caso2
Personal	\$0	\$60,000
Infraestructura	\$0	\$10,000
Herramientas	\$0	\$12,500
Total de Inversión	\$0	\$82,500
Desarrollo (Requerimientos, Diseño y Codificación)		
Defectos Encontrados	250	350
Costo de Corrección de Defectos	\$2,500	\$3,500
Pruebas		
Defectos Encontrados	0	600
Costo de Corrección de Defectos	0	\$60,000
Producción		
Defectos Encontrados	750	50
Costo de Corrección de Defectos	\$750,000	\$50,000
CoQ		
Conformidad	0	\$82,500
No-conformidad	\$752,500	\$113,500
Total	\$752,500	\$196,000
ROI	NA	74 %

Tabla 2.10: Análisis del ROI

$$ROI = \frac{752500 - 196000}{752500} = 0,74 = 74 \%$$

Este resultado nos quiere decir que se ahorró el 74 % del total de CoQ. Estos análisis son realizados comúnmente antes al finalizar las pruebas de sistema y antes de ingresar a producción. Por lo cual la situación del ROI puede empeorar conforme el sistema se encuentra en producción y los usuarios comienzan a encontrar los defectos mientras lo usan.

CAPÍTULO 3

Desarrollo del Trabajo

3.1 Concepto de Operaciones

El BM es un sistema de administración, de seguimiento y de mejora continua de la calidad personal y grupal en el desarrollo de sistemas de software. Esto lo realiza mediante el seguimiento de proyectos, seguimiento de actividades de desarrollo, seguimiento de actividades de calidad, administración de defectos y generación de estadísticas y métricas verdaderamente útiles para las organizaciones de software.

El BM tiene la flexibilidad necesaria para que la persona responsable dentro de la organización defina el ciclo de vida y las actividades por realizar dentro de un proyecto de software. Así mismo podrá definir la taxonomía a utilizar respecto a los diferentes defectos dentro del sistema, así como las plantillas a utilizar para las actividades de calidad. Esta flexibilidad permitirá que la aplicación pueda ser adoptada por un gran número de organizaciones de software con diferentes procesos de desarrollo de software.

El sistema estará basado en la tecnología web, por lo que no será necesaria la instalación física de la aplicación en los clientes que pretendan acceder a ella. Sólo será necesario el uso del navegador para acceder a este sistema y la instalación se hará en el servidor dedicado a la aplicación.

3.1.1 Objetivos

Los objetivos principales del sistema son:

- Lograr una mejora continua en el proceso de desarrollo de software, así como una mayor calidad en el producto final.
- Reducir el costo de implementar actividades de calidad dentro de la empresa.
- Proporcionar información valiosa a la empresa para la toma de decisiones respecto a cambios en sus procesos de desarrollo de software.
- Facilitar la evolución y adaptación de las diferentes actividades de aseguramiento de la calidad.
- Promover una cultura de calidad personal enfocada en la prevención de defectos.
- Proveer datos sobre el esfuerzo (costo) de las actividades de remoción de defectos.
- Facilitar un cambio cultural respecto a la forma de trabajar de pequeñas y medianas empresas.

3.1.2 Alcances

Se puede definir como alcance en cuanto a funcionalidad lo siguiente:

- Registrar y dar seguimiento a las actividades de desarrollo y calidad establecidas para el ciclo de vida de desarrollo.
- Hacer un seguimiento puntual a la inyección, remoción y corrección de defectos a lo largo de las diferentes etapas del ciclo de vida.
- Generar estadísticas y métricas de valor para la empresa y el personal en base a la información proporcionada por los usuarios del sistema.
- Dar una guía en los procedimientos principales de aseguramiento de la calidad.

3.1.3 Módulos del Sistema

El BM está dividido en 5 módulos representados en la figura 3.1:

- *Módulo de Administración.* Este módulo está encargado de realizar las altas, bajas y cambios de proyectos, actividades y usuarios.
- *Módulo de Actividades.* Este módulo está encargado de realizar la actualización de actividades y las tareas relacionadas con el seguimiento del proyecto.
- *Módulo de Calidad.* Este módulo está encargado de las plantillas para las actividades de remoción de defectos, así como el ciclo de vida para los proyectos.
- *Módulo de Defectos.* Este módulo está encargado del registro y seguimiento de defectos.
- *Módulo de Reportes.* Este módulo está encargado de la generación de reportes estadísticos personales, por proyecto, por equipo y de empresa.

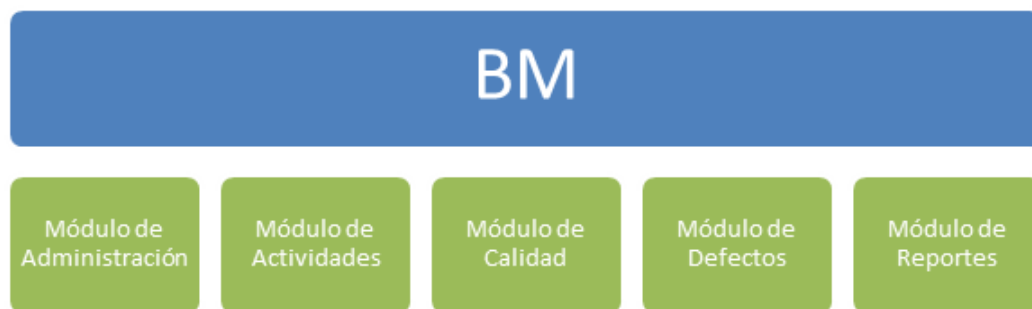


Figura 3.1: Módulos del BM

3.1.4 Tipos de Usuario

El BM tendrá tres tipos de usuarios: Administrador, Líder de Proyecto y Usuario. Están representados en la figura 3.2 que nos muestra los usuarios por nivel. Se representa en una pirámide invertida para denotar los privilegios y que los niveles superiores tienen toda la funcionalidad de los niveles inferiores. Entonces el Administrador puede utilizar la funcionalidad del Líder de Proyecto y del Usuario; mientras que el Líder de Proyecto tiene acceso a la funcionalidad de Usuario.

Sus características y funcionalidades principales son:

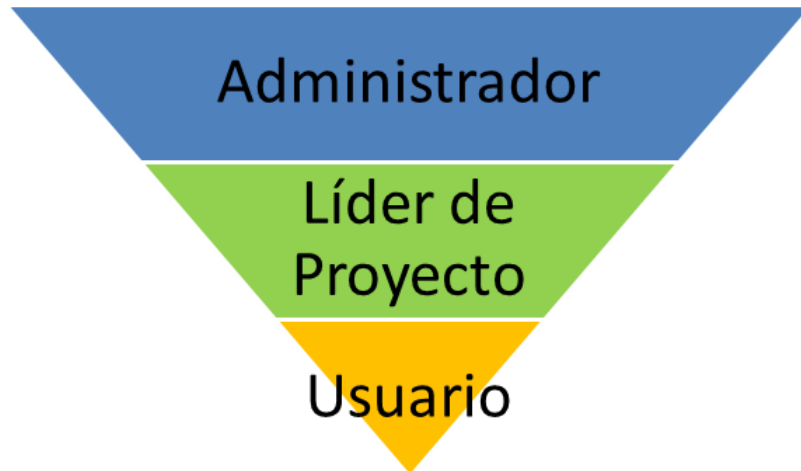


Figura 3.2: Tipos de usuario del BM

- *Administrador.* Es el tipo de usuario con más privilegios en el sistema y tiene responsabilidades y derechos a nivel organización, representa a la gerencia de la organización. Sus actividades principales son:
 - Alta, baja y modificación de proyectos.
 - Alta, baja y modificación de usuarios.
 - Asignación de recursos a proyectos.
 - Revisión de reportes y métricas por organización, proyecto e individuales.
- *Líder de Proyecto.* Es el tipo de usuario que le sigue en privilegios al Administrador y tiene responsabilidades y derechos a nivel de proyecto, representa al líder de uno o varios proyectos. Sus actividades principales son:
 - Definición de tipos de defecto.
 - Definición de ciclo de vida para el proyecto.
 - Definición de plantillas públicas para las actividades de calidad.
 - Definición de actividades de desarrollo y de actividades de calidad.
 - Asignación de las actividades a los usuarios.
 - Revisión de reportes y métricas a nivel proyecto e individuales.

- *Usuario.* Es el tipo de usuario con menos privilegios, representa un desarrollador del proyecto. Sus actividades principales son:
 - Seguimiento de las actividades asignadas.
 - Creación y modificación de plantillas de calidad propias.
 - Reporte de defectos.
 - Seguimiento de defectos asignados.
 - Revisión de reportes y métricas individuales.

Los detalles de las funcionalidades completas del sistema serán descritas en la sección [REFERENCIA].

3.1.5 Impacto

El BM impacta a los distintos niveles de cualquier organización dedicada al desarrollo de software.

- A nivel de alta gerencia:
 - Provocará una mayor formalidad en la manera de controlar y asignar recursos a nuevos y existentes proyectos.
 - Dará una visibilidad del costo de la calidad que tienen los distintos proyectos de software.
 - Permitirá conocer el esfuerzo real que toman los proyectos y hacer compromisos con más información en el futuro.
- A nivel de líder de proyecto:
 - Dará una mayor formalidad y disciplina para la realización de actividades de planeación referentes al ciclo de vida y a la calidad del producto.
 - Brindará una mejor visibilidad del estado actual del proyecto.
 - Ayudará a realizar una administración racional del proyecto.

- Facilitará la administración de la calidad del proyecto.
- A nivel de desarrollador se tiene el mayor impacto:
 - Implicará que los desarrolladores cuenten con la suficiente disciplina para realizar las actividades de calidad de la mejor manera posible.
 - Al registrar los defectos cometidos creará conciencia de estos dando un impacto inmediato a la calidad.
 - Ayudará a los desarrolladores a mejorar sus procesos personales de desarrollo.
 - Se generarán estadísticas y métricas con información verídica la cual permitirá analizar los procesos y mejorar las áreas más débiles.
 - En resumen, mejorará la forma en que los desarrolladores hacen su trabajo, haciendo que los productos que elaboren tengan una mayor calidad desde el inicio recortando el costo y el calendario de los proyectos.

3.1.6 Limitaciones

Dentro de las limitaciones identificadas para el sistema BM se encuentran:

- Si bien se permite el acceso a múltiples usuarios de manera simultánea, la concurrencia al momento de edición no está permitida.
- La generación de estadísticas y métricas se basa en la información y los datos introducidos por los distintos usuarios, por lo que en caso de que esta información no sea adecuada, las estadísticas y métricas generadas por el sistema tampoco lo serán.
- El acceso al sistema depende de la correcta operación de la red local de la empresa o del proveedor de servicios de Internet.
- La asignación de restricciones y privilegios sobre el uso de la aplicación para los diferentes tipos de usuarios está preestablecida, por lo que el cliente no podrá configurar estos permisos al momento de la instalación del sistema.

- Las estadísticas y métricas generadas por el sistema fueron determinadas con anterioridad, por lo que el cliente no tendrá la posibilidad de agregar, modificar o eliminar estadísticas o métricas.

3.2 Diseño del Sistema

El BM está construido con las siguientes tecnologías:

- Java 7 EE como lenguaje de propósito general.
- Spring como infraestructura para crear la aplicación web.
- MySQL como administrador de la base de datos.
- Velocity para realizar el scripting de las páginas web.
- JQuery para enriquecer la funcionalidad de las páginas web.

Con la combinación y uso de las tecnologías mencionadas se creó el BM. La arquitectura del sistema y el diseño de la base de datos serán presentadas en las secciones consecuentes.

3.2.1 Arquitectura

La arquitectura del BM se muestra en la figura 3.3. Los componentes de esta son los siguientes:

- *Vista*. En esta capa se encuentran los archivos que visualiza el usuario final. Estos archivos están contruidos en código HTML dinámicamente por Velocity y son enriquecidos con JQuery.
- *Controladores*. Es la capa de conexión entre la Vista y la Capa de Negocios. Está encargada de recibir las solicitudes de la Vista, llamar a la Capa de Negocios para realizar las operaciones y mandar los resultados de nuevo a la Vista.

- *Capa de Negocios*. Es la capa de conexión entre los Controladores y el Modelo. En esta capa está implementada la funcionalidad del sistema y contiene las operaciones por realizarse.
- *Modelo*. Es la capa de conexión entre la Capa de Negocios y del DAO. Esta capa es una representación de la base de datos. Es utilizada por la Capa de Negocios para crear los objetos, y obtiene los datos de estos objetos a través del DAO.
- *DAO*. El objeto de acceso a la base de datos (por sus siglas en Inglés DAO) es la capa que conecta la base de datos con el modelo. Esta capa se encarga de recibir solicitudes de información por parte del Modelo, obtiene la información de la base de datos y la regresa al modelo.
- *Base de Datos*. Es el contenedor que almacena la información de todo el BM.

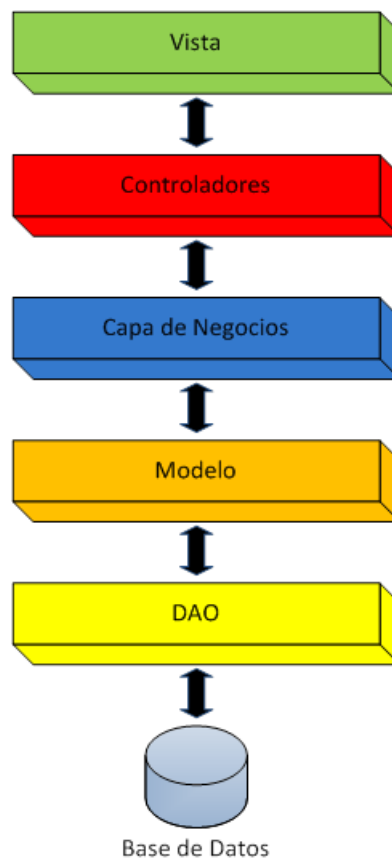


Figura 3.3: Arquitectura del BM

3.2.2 Base de Datos

La figura 3.4 nos muestra el diseño de la base de datos.

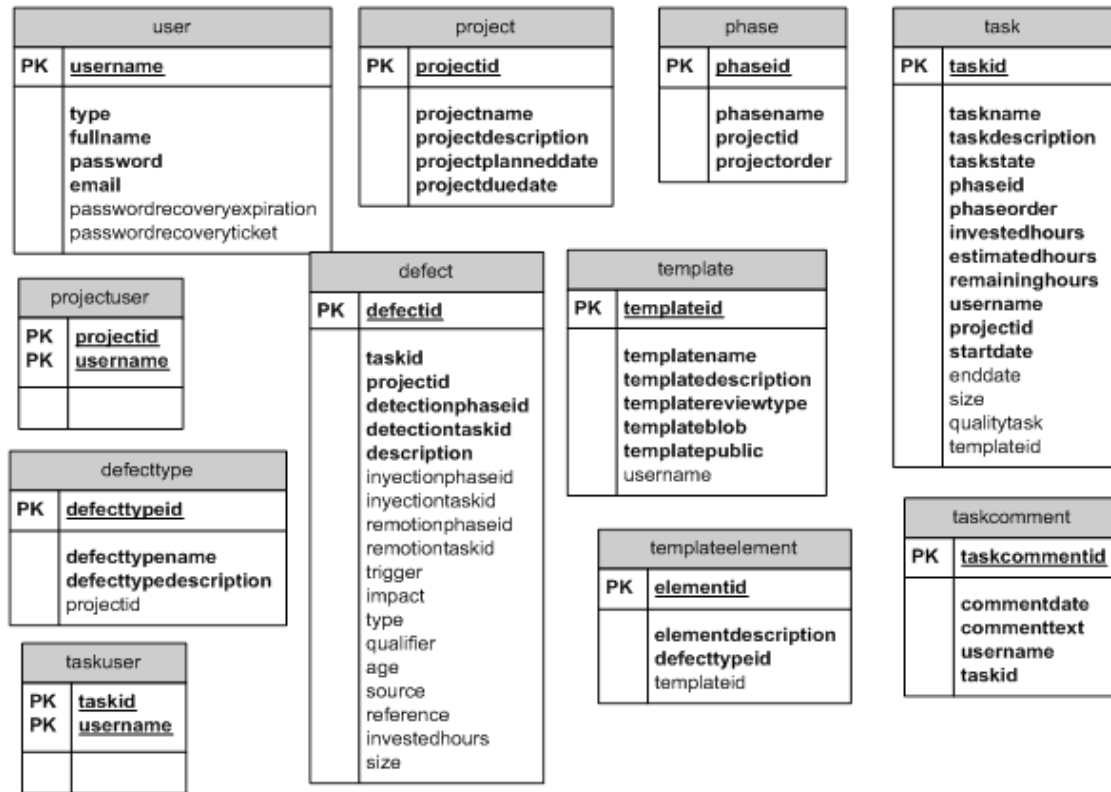


Figura 3.4: Base de Datos del BM

Esta contiene las siguientes tablas:

- *User*. Contiene los usuarios del sistema.
- *Project*. Contiene los proyectos del sistema.
- *Phase*. Contiene las fases del sistema. Cada fase pertenece a un proyecto en específico.
- *Task*. Contiene las tareas del sistema. Cada tarea está relacionada con un proyecto y con una fase del proyecto y puede ser una tarea de desarrollo o de calidad.
- *Taskcomment*. Contiene los comentarios realizados en las tareas. Se relaciona con una tarea.

- *Defect*. Contiene los defectos registrados en el sistema. Los defectos están relacionados con la tarea y la fase en la que fueron detectados.
- *Template*. Contiene las plantillas que sirven como guías para las actividades de calidad del sistema. Estas plantillas pueden ser públicas o privadas. Inicia con la plantilla para lenguajes de propósito general propuesta por Humphrey[1].
- *Templateelement*. Representa un elemento dentro de las plantillas de calidad, está relacionado a una plantilla en específico.
- *Defecttype*. Contiene los tipos de defectos del sistema. Inicia con los tipos de defectos del PSP[1].
- *Projectuser*. Es una tabla de soporte que ayuda a relacionar varios proyectos con varios usuarios.
- *Taskuser*. Es una tabla de soporte que ayuda a relacionar varios usuarios con varias tareas.

3.3 Funcionalidades del BM

En esta sección se describirán a detalle las secciones del BM. Encontraremos dos tipos de funcionalidades:

- *Funcionalidades de Administración*. Estas funciones son aquellas necesarias para el correcto funcionamiento del sistema, como puede ser administración de usuarios y de proyectos.
- *Funcionalidades de Valor Agregado*. Estas funciones son aquellas que ayudan a las empresas a obtener los beneficios mencionados en la sección 3.1.5. En estas se hará referencia a los objetivos que tienen y como son soportadas por la teoría.

3.3.1 Administración de Usuarios

Esta parte del sistema es meramente administrativa y solo los usuarios . Tiene las siguientes funcionalidades:

- *Alta de Usuarios.* Se registran nuevos usuarios especificando su nombre de usuario, nombre real, correo electrónico, privilegios y contraseña.
- *Modificación de Usuarios.* Se pueden editar los campos dados de alta en el Alta excepto la contraseña.
- *Cambio de Contraseña.* Existe una pantalla especial para el cambio de contraseña y recuperación de esta en caso de haberla perdido.
- *Eliminación de Usuarios.* Se pueden eliminar los usuarios siempre y cuando no sea el usuario \$admin\$ (usuario por default del sistema) y no esté enrolado en ningún proyecto.

3.3.2 Administración de Recursos

Otra parte del sistema administrativa pero con su grado de importancia. En esta sección un usuario con privilegios de administrador puede asignar a los diferentes usuarios del sistema a los proyectos existentes. Un usuario al ser asignado a un proyecto obtiene automáticamente la visibilidad de este.

3.3.3 Administración de Proyectos

Esta parte del sistema permite al administrador las siguientes funcionalidades:

- *Alta de Proyectos.* Se dan de alta nuevos proyectos especificando nombre del proyecto, descripción breve, fecha de entrega planeada y fecha de entrega real. Después de dar de alta un proyecto se pasa a la creación de su ciclo de vida.
- *Modificación de Proyectos.* Se pueden modificar los datos de los proyectos registrados en su alta. Aparte se especifica la fase actual en la que se encuentra el proyecto.

- *Eliminación de Proyectos.* Los proyectos pueden ser eliminados solamente cuando no tengan información registrada dentro de estos.

3.3.4 Ciclo de Vida de Proyectos

El BM permite al administrador o al líder de proyecto la creación de un ciclo de vida propio para el proyecto, o la selección de uno predefinido que puede ser:

- *Cascada.* Es el ciclo de vida más clásico de los proyectos de desarrollo de software. Tiene las fases de Requerimientos, Diseño, Codificación, Pruebas y Mantenimiento.
- *Iterativo.* Es el ciclo de vida preferido por los proyectos desarrollados en empresas con filosofías ágiles. Este ciclo es una variación del ciclo de vida de cascada, pero en vez de realizar el diseño, la codificación y las pruebas completas en una sola ocasión, dividen estas tres tareas en varias iteraciones para enfrentar los posibles cambios.

La figura 3.5 nos muestra la pantalla donde se permite elegir los ciclos de vida. Una vez seleccionado un ciclo de vida default o comenzando a crear uno propio se pueden agregar, modificar o eliminar las fases existentes como se muestra en la figura 3.6. Una fase no puede ser eliminada una vez que tenga actividades registradas. Los datos que contiene una fase de un proyecto son: Nombre, tipo, descripción y orden en el proyecto.

Los tipos de fase predefinidos en el BM son los siguientes:

- *Requerimientos.* Esta es la fase donde se realiza la licitación y el análisis de los requerimientos del proyecto.
- *Diseño.* En esta fase se toman las decisiones más importantes respecto a como será construido el proyecto. Lo que se realice en esta fase determinará como es que el proyecto funcionará y si tendrá éxito o no. Los productos de trabajo de esta fase más comunes son la arquitectura y el diseño detallado.
- *Codificación.* En esta fase se construye el sistema, también es llamada fase de construcción o programación. Es la etapa de programación y la primera que nos viene a la mente en el desarrollo de software.

- *Revisión.* Esta fase puede ser utilizada como comodín. Se incluyó en caso de que el equipo de desarrollo decida agregar alguna fase completa de revisión de los productos de otra fase. Ya que el costo de los errores aumenta exponencialmente conforme avanza de fases el proyecto[21], entonces es muy importante evitar la inyección de errores en fases tempranas como Requerimientos y Diseño, así que podría incluirse una fase completa para realizar actividades como inspecciones al diseño y a los requerimientos. En caso de que se decida no utilizar una fase completa para revisiones, es muy importante que al menos se realicen revisiones personales al trabajo realizado.
- *Pruebas.* Esta etapa corresponde a las pruebas clásicas. Se recomienda al menos la realización de pruebas unitarias y de sistema.
- *Mantenimiento.* Esta fase es cuando el sistema ha salido a producción y se continúan agregando nuevas funcionalidades o defectos encontrados por el cliente. El objetivo del BM es que no haya ningún defecto reportando por el cliente.

The image shows two side-by-side web forms. The left form, titled 'Agregar Fase', has a light blue background and contains the following fields: 'Nombre de la fase:' with a text input, 'Tipo:' with a dropdown menu showing 'REQUIREMENTS', 'Descripción:' with a large text area, and 'Después de:' with a dropdown menu. At the bottom are 'Cancelar' and 'Aceptar' buttons. The right form, titled 'Ciclo de Vida Default', has a light blue background and contains 'Tipo de ciclo de vida' with a dropdown menu showing 'Cascada', 'Cascada', and 'Iterativo'. A red rectangle highlights this dropdown menu. Below the dropdown are 'Cancelar' and 'Aceptar' buttons. The 'Aceptar' button has a blue checkmark icon.

Figura 3.5: Ciclo de Vida Default

Es importante destacar que la definición del ciclo de vida es el primer paso para iniciar la planeación del proyecto. La planeación es una de las prácticas recomendadas por [2] para realizar la administración racional. Si una organización quiere tener éxito en el negocio del software es clave la planeación del proyecto.

Agregar Fase

Nombre de la fase:

Tipo: REQUIREMENTS ▾

Descripción:

Después de: ▾

[Cancelar](#)
[Aceptar](#)

Fases Actuales

FASE	TIPO	DESCRIPCIÓN	ACCIÓN
Requerimientos	REQUIREMENTS	Licitación y análisis de requerimientos	Borrar
Revisión de Requerimientos	REVIEW	Revisión del documento de requerimientos	Borrar
Diseño	DESIGN	Diseño y arquitectura del sistema	Borrar
Revisión de Diseño	REVIEW	Revisión del documento de diseño	Borrar
Codificación	CODING	Construcción del sistema	Borrar
Revisión de la Codificación	REVIEW	Revisión de la construcción del sistema	Borrar
Pruebas	TESTING	Pruebas del sistema	Borrar
Mantenimiento	MAINTENANCE	Mantenimiento del sistema	Borrar

Figura 3.6: Edición del Ciclo de Vida

Para el uso correcto del BM y obtener su mayor potencial se recomienda una definición a conciencia del ciclo de vida. Una vez definido el ciclo de vida se puede pasar a la definición de las tareas dentro de cada fase. También es de suma importancia que cada una de estas tareas cuente con su actividad de revisión, para fomentar la cultura de la prevención antes de las pruebas.

3.3.5 Administración y Seguimiento de Actividades

El registro, actualización y medición de las actividades realizadas dentro de un proyecto es clave para hacer un trabajo de calidad. Lo que es medido es administrado, y lo que es administrado se hace correctamente, en cambio, lo que no es medido no se administra y por lo tanto no se termina[2].

El BM facilita la labor de la planeación y administración de actividades por medio de distintas funcionalidades:

- La creación de un ciclo de vida para el proyecto como se explicó en la sección [REFERENCIA].

- El alta, baja y modificación de actividades.
- El seguimiento y actualización de las actividades.

El BM tiene dos tipos de actividades:

- *Actividades de Desarrollo.* En esta categoría caen todas las actividades relacionadas con el proceso de desarrollo de software en las cuales se generan productos de trabajo. Ejemplos de estas son: Licitación de requerimientos, diseño detallado del módulo de un sistema, programación, pruebas unitarias entre otras. Este tipo de actividades tienen subtipos, estos son los siguientes:
 - *REQUIREMENTS.* Actividades relacionadas a una fase de requerimientos.
 - *DESIGN.* Actividades relacionadas a una fase de diseño.
 - *DEVELOPMENT.* Actividades relacionadas a la fase de construcción.
 - *TESTING.* Actividades relacionadas con alguno de los tipos de prueba mencionados en la sección [REFERENCIA].
- *Actividades de Calidad.* Son las actividades de prevención y evaluación realizadas en el proyecto de desarrollo las cuales nos ayudan a evitar la inyección de defectos o a detectarlos lo antes posible dentro del ciclo de desarrollo. Las actividades de calidad que maneja el BM son las siguientes:
 - *PERSONAL REVIEW.* Actividades donde se realiza una revisión personal a un producto de trabajo [REFERENCIA].
 - *PEER REVIEW.* Actividades donde se realiza una revisión entre colegas de un producto de trabajo [REFERENCIA].
 - *WALKTHROUGH.* Actividades donde se realiza una caminata a un producto de trabajo [REFERENCIA].
 - *INSPECTION.* Actividades donde se realiza una inspección a un producto de trabajo [REFERENCIA].

Para obtener mayores beneficios del BM y tener un nivel óptimo de calidad se recomienda la siguiente forma de trabajo:

- Realizar al menos una revisión personal a cada producto de trabajo utilizando una plantilla de calidad. Por ejemplo cada que el desarrollador termine de programar un clase del sistema, realizar una revisión personal de esta apoyándose con la plantilla de calidad default o una elaborada personalmente.
- Realizar una inspección a cada producto mayor de trabajo. Un producto mayor de trabajo es un producto que representa el cierre de una fase, por ejemplo: El documento de requerimientos al terminar la fase de requerimientos, la arquitectura del sistema al terminar el diseño conceptual, entre otros.

Las funcionalidades básicas con las actividades tanto de desarrollo como de calidad son las siguientes:

- *Alta de actividades.* Se registra una nueva actividad en el sistema con los siguientes datos: Nombre de la tarea, tipo, fase, descripción, esfuerzo planeado, reponsable, fecha de inicio y fecha meta. La forma se puede ver en la figura 3.7.
- *Modificación de actividades.* Se modifican los datos con los cuales se dieron de alta las actividades.
- *Baja de actividades.* Una actividad puede ser dada de baja solo en el caso de que no tenga registrado esfuerzo.

El seguimiento de las actividades se muestra en la figura 3.8. Esto nos permitirá hacer un seguimiento y administración adecuado de cada tarea. Se recomienda que diariamente el responsable de la actividad haga una actualización de esta con las siguientes consideraciones:

- Escribir en el campo de Agregar Esfuerzo el número de horas reales que le dedicó a la tarea. Por ejemplo: Un desarrollador tiene como tarea programar la interfaz de un sistema un día de trabajo; pero en el día de 8 horas, pasó 2 horas en juntas, otra hora

El formulario 'Nueva Tarea' tiene un fondo gris claro. En el centro hay un recuadro con fondo azul claro que contiene los campos de entrada. El título 'Nueva Tarea' está en la parte superior del recuadro en un color morado. Los campos incluyen: 'Nombre de la Tarea' (campo de texto), 'Tipo' (menú desplegable con 'REQUIREMENTS' seleccionado), 'Fase' (menú desplegable con 'Code' seleccionado), 'Descripción' (área de texto grande), 'Esfuerzo Planeado' (campo de texto), 'Responsable' (menú desplegable con '-- Seleccionar --' seleccionado), 'Fecha de Inicio' (campo de texto), 'Fecha Meta' (campo de texto), un botón 'Cancelar' con un subrayado, y un botón 'Aceptar' con un icono de checkmark verde.

Figura 3.7: Creación de Actividades

revisando el correo electrónico y una hora más en descansos, entonces ese día debe reportar 4 horas al esfuerzo y no las 8 horas del día. Esto es muy importante para que las empresas identifiquen las horas reales de trabajo que tienen los desarrolladores y así puedan hacer más eficiente el tiempo en la oficina.

- Si el esfuerzo restante se deja en 0 la actividad será marcada como terminada, así que es importante que se estime el esfuerzo restante y se coloque en el campo respectivo. Esto ayudará a los desarrolladores a mejorar sus habilidades de estimación y generar datos históricos.
- Colocar el tamaño de la tarea en las unidades que maneje la empresa. Para el uso del BM se recomienda utilizar LOC para los programas por su facilidad al momento de calcular y lo común que es dentro de la industria, sin embargo se pueden utilizar otras métricas como puntos de función. Esto también ayudará a crear datos históricos y facilitará el dimensionamiento de futuros proyectos.

- Agregar comentarios para cada suceso importante que surja en las actividades. Los comentarios quedarán registrados y ser consultados después.
- Tener en cuenta que la Fecha Fin no es la fecha en que se finalizó la tarea, si no la fecha en que estaba planeado en que se finalizara. La fecha cuando se finaliza la tarea se registra de forma automática cuando el estatus de la tarea pasa a COMPLETADA.

Ver/Modificar Tarea

Nombre:

Tipo:

Fase:

Descripción:

Responsable:

Esfuerzo Planeado:

Esfuerzo Acumulado:

Agregar Esfuerzo:

Esfuerzo Restante:

Tamaño de la Tarea:

Estatus: COMPLETADA

Fecha de Inicio:

Fecha de Fin:

Comentarios:

Agregar Comentario:

[Cancelar](#)

Figura 3.8: Seguimiento de Actividades

3.3.6 Administración y Seguimiento de Defectos

Bibliografía

- [1] W. S. Humphrey, *PSP A Self-Improvement Process for Software Engineers*. Addison - Wesley, 2005.
- [2] ———, *Winning with Software, An Executive Strategy, How to Transform Your Software Group into a Competitive Asset*. Addison - Wesley, 2002.
- [3] P. B. Crosby, *Quality Is Free: The Art of Making Quality Certain: How to Manage Quality - So That It Becomes A Source of Profit for Your Business*. McGraw-Hill Companies, 1979.
- [4] J. M. Juran and F. M. Gryna, *Juran's quality control handbook*. McGraw-Hill, 1988.
- [5] E. Davies and M. Whyman, "Iso 9000:2000-new iso, new responsibilities for top management," *Engineering Management Journal*, pp. 244 – 248, 2000.
- [6] D. Stelzer, W. Mellis, and G. Herzwurm, "Software process improvement via iso 9000? results of two surveys among european software houses," in *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, 1996, pp. 703 – 712.
- [7] C. Yoo, J. Yoon, B. Lee, C. Lee, J. Lee, S. Hyun, and C. Wu, "An integrated model of iso 9001:2000 and cmmi for iso registered organizations," in *Software Engineering Conference, 2004. 11th Asia-Pacific*, 2005, pp. 150–157.
- [8] J. E. Bentley and W. Bank, "Software testing fundamentals—concepts, roles, and terminology."
- [9] S. McConnell, *Professional Software Development*. Addison - Wesley, 2004.

- [10] H. Remus and S. Ziles, "Prediction and management of program quality," in *Proceedings of the Fourth International Conference on Software Engineering, Munich, Germany*, 1979, pp. 341–350.
- [11] M. Bush, "Improving software quality: The use of formal inspections at the jet propulsion laboratory," in *Twelfth International Conference on Software Engineering*, March 1990, pp. 196–199.
- [12] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski, "Software inspections: An effective verification process," *IEEE Software*, vol. 8, pp. 31–36, 1989.
- [13] B. Ragland, "Inspections are needed now more than ever," *Journal of Defense Software Engineering*, vol. 38, November 1992.
- [14] G. W. Rusell, "Experience with inspections in ultralarge-scale developments," *IEEE Software*, pp. 25–31, January 1991.
- [15] M. L. Shooman and M. I. Bolsky, "Types, distribution, and test and correction times for programming errors," in *Proceedings of the 1975 Conference on Reliable Software*. IEEE, 1975.
- [16] E. F. Weller, "Lessons learned from two years of inspection data," *IEEE Software*, pp. 38–45, 1993.
- [17] K. Owens, "Software detailed technical reviews: Finding and using defects," in *Proceedings Wescon*, 1997.
- [18] L. Harjumma, "Peer reviews in real life - motivators and demotivators." *International Conference on Quality Software*, 2005.
- [19] G. M. Freedman and D. P. Weinberg, "Reviews, walkthroughs and inspections," *IEEE Transactions on Software Engineering*, 1984.
- [20] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on Software Engineering*, vol. 18, pp. 943–956, 1992.

-
- [21] L. Lazic, A. Kolasinac, and D. Avdic, “The software quality economics model for software project optimization,” *WSEAS Transactions on Computers*, vol. 8, pp. 21–47, 2009.
 - [22] J. Capers, *Estimating Software Costs*. McGraw-Hill, 2007.
 - [23] B. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.
 - [24] R. Black, *Managing the Testing Process*, segunda edición ed., Wiley, Ed. Wiley, 2002.
 - [25] D. Houston and B. Keats, “Cost of software quality: A means of promoting software process improvement,” *Quality Engineering*, vol. 10, pp. 563–573, 1998.
 - [26] J. Juran and A. B. Godfrey, *Juran’s Quality Handbook*. McGraw-Hill, 1998.
 - [27] S. T. Knox, “Modeling the cost of software quality,” *Digital Technical Journal*, vol. 5, pp. 9–16, 1993.