Las Inspecciones de Software y las Listas de Comprobación

Tesis presentada en opción al
título de Master en Informática Aplicada a la Ingeniería y la Arquitectura

Autor: Ing. MsC. Roberto Félix Zamuriano Sotés
rzamuriano@gmail.com
rzamurianos@netvalle.univalle.edu

ÍNDICE

INDICE DE TABLAS

INDICE DE FIGURAS

INTRODUCCIÓN

A medida que la Tecnología de la Información va desarrollándose, los problemas van siendo más complejos, esto obliga a buscar nuevas soluciones, nuevos caminos o nuevos paradigmas que solucionen los problemas. La solución generalmente, incluye un software, por la gran cantidad de información y la complejidad del problema. Pero, el desarrollo del Software se ha convertido en una tarea muy compleja que ha sobrepasado en gran medida la habilidad para el mantenimiento de las empresas que se dedican al desarrollo de software.
Hoy en día, las empresas cubanas y al igual que las del mundo, buscan una alternativa para mejorar la producción de software, garantizar la calidad y la satisfacción del usuario. El aumento de la cultura hacia la excelencia y la administración del desarrollo, darán como resultado la mejor producción y empleo de los recursos para la fabricación.

Cuba es un país con mucho conocimiento, creatividad y posibilidades para lograr un gran avance en desarrollo de software y competir en el mundo. Es muy claro que para lograr ser parte de la competencia, debe iniciar el camino del mejoramiento del proceso de desarrollo del software. En el trabajo, se da un modelo de inspección de software y una herramienta que lo automatiza llamada SPIS (Soporte al Proceso de Inspecciones de Software que se basa en la utilización de las Listas de Comprobación), ayudando de esta forma al Aseguramiento de Calidad de Software y a las personas que realizan esta actividad en la entidad.

En el transcurso del desarrollo del Proceso de Software existen modelos para llevar a cabo su planificación, ejecución, desarrollo, aceptación, culminación e implantación. Estos modelos, en el mundo tienen un fin que es la “calidad para el cliente”, para todo producto conocido y más aun para el software. Para llegar a un producto de calidad y satisfacer la exigencia y requisitos de los clientes se sigue un proceso predeterminado y estudiado, el cual es la base para elaborar diferentes proyectos de acuerdo con los requisitos del cliente.

Al iniciar un proyecto se encuentran muchos requisitos; se trata de hallar soluciones que los cumplan o solucionen. Al encontrar las soluciones, es muy necesario valorar la

calidad de cada una de estas, ver si los objetivos del proyecto, con estas nuevas soluciones, se cumplen. Desde este momento, se inicia el problema de la calidad, lo cual, implica cumplir un riguroso modelo del proceso de desarrollo, lo cual implica utilizar un complejo conjunto de estándares, técnicas y métodos. Para cada tipo de empresa, ya sea de materias, de servicio o de productos intangibles como el software es muy necesario tener un modelo de proceso establecido que ayude a controlar la calidad del producto que se pone a consideración del usuario o cliente.

Generalmente en las empresas o personas que desarrollan software de cualquier tipo no cumplen los modelos de procesos, para la realización del software, algunas veces estos modelos son desconocidos. Por este motivo se incurren en errores de diferentes tipos dando lugar al incremento de los costos, lo cual conlleva a una vida muy corta del software.

Dentro de la empresa cubana existen problemas, los cuales llevan a la producción de software con una baja calidad. Por ejemplo: La falta de conocimiento, por parte de los directivos de la industria informática, de los distintos enfoques empresariales que se relacionan con la producción de software; la ausencia de una coordinación a nivel nacional que rija la producción de software de forma cooperada; la ausencia de planes de desarrollo que permitan que los productos que salgan al mercado tengan una buena relación de prestaciones de acuerdo a las normas internacionales de calidad; la no identificación de las áreas adecuadas para la creación de software; la ausencia de una cultura de producción de software en la que se realicen estudios por parte de equipos multidiciplinados encaminados a la creación de un producto orientado a un mercado específico, todo esto cumpliendo con los parámetros de calidad y la culminación en tiempo; y por último, la mala calidad de muchos de los productos de software que se realizan en el país [68].
Por otro lado, dentro del Instituto Superior Politécnico José Antonio Echeverría (ISPJAE) y el Centro de Referencia de Ingeniería de Software (CRIS) se han detectado los siguientes problemas, al realizar un estudio de calidad dentro de las empresas cubanas, son los siguientes:

- La ausencia de información acumulada sobre el tiempo dedicado a determinadas tareas, hace que sea muy difícil estimar con relativa precisión la fecha de terminación.
- No existe una formalización o estandarización del tiempo dedicado a una tarea, ni del proceso de control del trabajo.
- La ausencia de procedimientos que permitan planificar y controlar el proceso de software.
- La ausencia de mecanismos que permitan medir el trabajo realizado por los especialistas.
- No se utiliza una estructura organizativa adecuada que apoye el desarrollo y mantenimiento del software.
- La ausencia de condiciones para el trabajo en equipos.
- La no existencia o la poca documentación sobre el análisis, el diseño y el desarrollo del software.
- La carencia de un mecanismo propio para el control de versiones.
- No existe un grupo dedicado a la atención al cliente, de forma que no sean los desarrolladores quienes atiendan las solicitudes o preguntas de los clientes que no están relacionadas con ellos.

El proceso de software que garantice la calidad adecuada para cualquier institución, se define de la siguiente manera:

“Es un conjunto de actividades, métodos, prácticas y transformaciones que las personas usan para desarrollar y mantener software y sus productos asociados” [1] [12].

Cuando una empresa que desarrolla software crece, necesita ampliar su organización, para esto, algunas veces, utilizará un modelo establecido internacionalmente para llevar a cabo las actividades o funciones que le permitan cumplir sus metas y objetivos con una buena calidad hacia los clientes o usuarios. Dentro de este modelo de proceso existe un Plan de Aseguramiento de Calidad y se define como un conjunto de personas que están encargadas

de llevar el seguimiento, al modelo de proceso de desarrollo, dentro de la empresa. Este plan y las personas que están a su cargo son muy importantes para la empresa.

Lastimosamente, la mayoría de las empresas de software hoy en día no establecen normas, ni objetivos, ni metas, ni políticas adecuadas para lograr que los productos elaborados cumplan; primero, normas internacionales para llevar a cabo el mantenimiento o para la exportación; segundo, la organización mejore incrementalmente su conocimiento y el dominio completo, entre los desarrolladores, sobre el proceso de software; tercero, la tecnología utilizada sea explotada al máximo; y cuarto, se tenga un grupo de personas que continuamente mejoren el proceso y velen por la calidad en la vida del proyecto.

Por estas razones, han surgido modelos del proceso de software, que se explican brevemente a continuación. Estos modelos se utilizan en el desarrollo de proyectos de software. Cada uno de estos establece la madurez por niveles que va incrementándose a medida que la empresa va cumpliendo los requisitos de entrega en tiempo de los proyectos, mejore la capacidad de administración, alcance un conocimiento continuo entre los desarrolladores actuales y nuevos sobre el proceso de software, se garantice que las actividades internas correspondan a procesos planificados, sean usables y consistentes las actividades, los procesos se actualicen y mejoren, existan pruebas pilotos, se realice un análisis de costo y beneficio para el cliente, y los roles y responsabilidades estén claros en los procesos y en la organización.

- El Conjunto de estándares ISO-15504 SPICE (Software Process Improvement and Capability Determination). Desarrollado por el WG10 de la ISO (Internacional Organization for Standardization) [59].
- ISO 9000 – 9001. La Organización Internacional para la Estandarización, mejor conocida como ISO, promueve la estandarización internacional, de tal manera que se facilite el intercambio de bienes y servicios así como el desarrollo científico y tecnológico mediante el comité técnico TC176, encargado de la normalización del Aseguramiento y Administración de la Calidad. ISO 9000-3 es

una guía y no una norma, que describe los elementos de un sistema de calidad orientado al software [60].

- Tick-It. Es una iniciativa de la industria del Reino Unido, que consiste en un esquema de certificación basado en la norma ISO 9000, guía ISO9000-3. Pone énfasis especial en la experiencia y en la capacitación que deben tener los auditores calificados, tanto en el área de desarrollo de software como en la evaluación de los criterios específicos de la norma. Tick-It es un esquema más riguroso que ISO 9000 [61].
- El Modelo de Madurez de la Capacidad (CMM: Capability Maturity Model). Desarrollado por el Instituto de Ingeniería de Software (SEI) de la Universidad Carnegie – Mellon en USA. Se basa en cinco niveles de madurez de las empresas que producen software, los niveles se enfocan en los procesos de desarrollo, capacidad de los procesos y su desempeño. [56]
- El Modelo de Madurez de Capacidad Integrado (CMMI: Capabily Maturity Model Integration). El Proyecto CMMI es un esfuerzo cooperativo del Departamento de Defensa de los Estados Unidos y el Instituto de Ingeniería de Software (SEI). El propósito del proyecto es proveer mejoras en costo, cumplimiento de cronogramas y calidad de los proyectos eliminando la complejidad de múltiples modelos de CMM. [50] [54]

Ya que el trabajo esta dirigido para la introducción en las empresas de los modelos CMM o CMMI, se explica brevemente; las diferencias entres estos dos modelos para luego señalar los niveles de madurez. También, durante el trabajo, se comprende como juega un papel importante las Inspecciones de Software con las Listas de Comprobación en los distintos niveles de madurez para los dos modelos. Hay que señalar, “que la aplicabilidad de los modelos para la mejora de procesos depende de la situación y del tipo de cada organización, ya que los programas de mejora deben estar basados en la misión y los objetivos del negocio que tenga cada organización.” [52]

La introducción de un modelo de mejora es necesario en la empresa, ya que uno de los requisitos a nivel internacional para ser subcontratado y desarrollar software, es estar en el

nivel 3 o nivel 2 de CMM o CMMI, o tener una certificación ISO, por esta razón la India ha tenido un gran auge en este sentido.

CARACTERÍSTICAS DE MODELO DE CAPACIDAD DE MADUREZ (CAPABILITY MATURITY MODEL – CMM)

CMM persigue la evaluación de los procesos de desarrollo de software dentro de una organización proponiendo un plan de mejoramiento en base a una serie de niveles que van desde un proceso inmaduro hasta un proceso disciplinado, ordenado y de mejoramiento continuo.

La madurez de un proceso de software, determina en que grado un proceso de software es explícitamente definido, administrado, medido, controlado y hecho efectivo. La madurez es un indicador de la capacidad del proceso de software para lograr sus objetivos y resultados esperados [57].

Una Empresa logra mayor madurez mediante la institucionalización del proceso de desarrollo de software, estableciendo las políticas, estándares y estructuras organizativas.

CMM esta conformado por niveles de madurez, que indican la capacidad del proceso. Los niveles, contienen áreas claves de proceso (PKS), con la ayuda de ellas se alcanza los objetivos. Las áreas están organizadas con características comunes, que se aplican a la implementación o institucionalización. Las características comunes entre las áreas contienen prácticas claves, que describen la infraestructura o las actividades. [5][56]

Nivel 1 (Inicial):
No hay proceso definido para el proceso de Software.

Nivel 2 (Repetible).

Gestión del **proceso** de **software**: coste, planificación y funcionalidad, sigue un conjunto de áreas.

- Administración de Requisitos.

- Planificación de Proyectos de Software
- Seguimiento y supervisión de Proyectos de Software
- Administración de Subcontratos de Software
- Aseguramiento de Calidad de Software (SQA)
- Gestión de Configuración del Software

Nivel 3 (Definido).

Desarrollo y **Mantenimiento documentado** y **Estandarizado**. Definido, los procesos son estándares en toda la empresa y se valida si éstos son los adecuados. El conjunto de áreas es el siguiente:

- Enfoque del Proceso de la Organización
- Definición del Proceso de la Organización
- Programa de Formación
- Gestión de Integración del Software
- Ingeniería del Producto de Software
- Coordinación entre grupos
- Revisiones periódicas.

Nivel 4 (Gestionado).

Medidas del **Producto y del Proceso**. Registro de valores de Calidad.

Administración, se establecen métricas de cada una de las partes del proceso y se analizan sus resultados. Las áreas son las siguientes:

- Gestión cuantitativa del Proceso
- Gestión de Calidad del Software

Nivel 5 (Optimizado).

Resultados cuantificados, con opción de mejora

Los procesos se optimizan continuamente y se definen estándares.

Las áreas son las siguientes:

- Prevención de Defectos
- Gestión de la Tecnología
- Gestión de Cambios en el Proceso

El modelo CMM ofrece una estrategia para mejorar el proceso, junto con un marco para evaluar empresas. Mejorar el proceso es establecer un rango de calidad, es una medida para otros de cómo es llevado el proceso de Software, de esta forma, durante el avance entre los niveles se va robusteciendo la fabricación de software.

CARACTERISTICAS DEL MODELO DE CAPACIDAD DE MADUREZ INTEGRADO (CAPABILITY MATURITY MODEL INTEGRATION – CMMI)

En Diciembre del 2001, el Instituto de Ingeniería de Software (SEI) lanzo la versión 1.1 del CMMI es una nueva versión de CMM. CMMI es un modelo de Procesos, o una colección estructurada de componentes que describen las características de los procesos efectivos de software que han sido probados por la experiencia.
También, esta dividido por Áreas de Proceso como CMM, cada área está compuesta por metas, cada una de estas metas está conformada por prácticas. Las metas y las prácticas son específicas o genéricas. Las prácticas corresponden a una sola Área de Proceso, mientras que las áreas específicas van a través de todas las áreas de proceso.
Las metas y prácticas genéricas, tienen elaboraciones que las inician para cada área de proceso.

Las metas específicas se aplican a solo un área de proceso, mientras que las metas genéricas se aplican a más de un área de proceso. Una práctica específica es una actividad considerada importante para alcanzar una meta específica. Las prácticas genéricas son prácticas que se aplican a cualquier área de proceso, y que pueden mejorar el desempeño y control de cualquier proceso.

Existen dos representaciones para este Modelo: por etapas (Staged) [62] y continua (Continuous) [63]. La representación continua es claramente más flexible en cuanto a que

permite formar una estrategia de mejora que se adapte a las metas globales de la respectiva organización. La representación por etapas, en contraste, es el modelo preferido por organizaciones que quieren migrar más fácilmente de CMM a CMMI.

Los niveles de CMMI a CMM tienen la misma definición.
Nivel 1 (Inicial):
No hay proceso definido para el proceso de Software.

Nivel 2 (Administrado):

- Administración de Requisitos.
- Planificación de Proyecto.
- Monitoreo y Control del Proyecto.
- Administración del Acuerdo del Proveedor.
- Medición y Análisis.
- Aseguramiento de Calidad del Proceso y el Producto.
- Administración de Configuración.

Nivel 3 (Definido)

- Desarrollo de Requisitos.
- Solución Técnica.
- Integración del Producto.
- Verificación.
- Validación.
- Enfoque del Proceso Organizativo.
- Definición del Proceso Organizativo.
- Entrenamiento Organizativo.
- Administración integrada del Proyecto.
- Administración de Riesgos
- Análisis de Decisiones y Resolución.
- Ambiente Organizativo para la Integración [62].
- Equipo Integrado [62].

Nivel 4 (Administrado Cuantitativamente):

- Desempeño Funcionamiento del Proceso Organizativo.
- Administración Cuantitativa del Proyecto

Nivel 5 (Optimizado):

- Innovación y Despliegue Organizativo.
- Análisis de Causas y Resolución

Cada uno, CMM y CMMI, establece un punto de partida para encontrar las soluciones a los requisitos y problemas que pueda tener una empresa de desarrollo software, logrando con estos modelos a cumplir con sus metas y objetivos hacia el cliente.

Sin embargo, de estos modelos descritos anteriormente en un artículo publicado en Fortune se dice: [2]

"Un estudio más reciente del Standish Group hecho sobre 352 compañías de software, donde se estudiaron más de 8.000 proyectos de software, revelaron lo siguiente:

· El 31% de todos los proyectos de software fueron cancelados antes de terminarse ($81 billones de dólares norteamericanos perdidos).
· El 53% de los proyectos tuvieron un costo 189% mayor de lo estimado.
· El 9% de los proyectos se terminaron a tiempo y dentro del presupuesto (compañías grandes).
· El 16% de los proyectos se terminaron a tiempo y dentro del presupuesto (compañías pequeñas).

A raíz de estos datos, se les preguntó a las empresas sobre las causas de estos problemas. Las tres principales razones expuestas fueron las siguientes:

· Falta de información por parte de los usuarios (12.8%)
· Especificaciones y requisitos incompletos (12.3%)
· Cambios en las especificaciones y requisitos (11.8%)

Este estudio muestra que los defectos cometidos durante la fase de requisitos son extremadamente caros de reparar. En proyectos grandes, este tipo de defectos son muy frecuente. En el estudio de un proyecto de la fuerza aérea de USA, los defectos fueron clasificados según la fuente de donde provenían. Se encontró que los defectos de la etapa de requisitos comprendían el 41% del total de defectos, mientras que los defectos en la lógica del diseño comprendían solamente el 28% del total [2].

El problema radica en el Proceso de Desarrollo del Software, ya que no existe un método para la detección de defectos, que es la parte esencial para el inicio de la mejora continua en el proceso de desarrollo. La Inspección con las Listas de Comprobación es necesaria para iniciar la mejora en el proceso de desarrollo, la cual, es una herramienta alternativa, haciendo que se detecten, la mayoría de los defectos a tiempo. Además, sin realizar una planificación y seguimiento para llevar a cabo esta actividad no es posible sistematizarla.

Por tanto, el objetivo general será: Permitir que el Grupo de Aseguramiento de Calidad obtenga un modelo de Inspección de Software que ayude a controlar la calidad durante el proceso de software, a través de una herramienta automatizada que usa las listas de comprobación.

Teniendo como objetivos específicos los siguientes:

- Establecer un modelo de inspección de software de acuerdo a la realidad que nos rodea.
- Establecer una clasificación de las listas de comprobación de acuerdo a las fases de un proyecto y estándares internacionales.
- Proponer pasos para las inspecciones de Software, haciendo uso de las listas de comprobación.
- Desarrollar una aplicación automatizada que refleje el modelo de inspección de software a establecer. El cual debe ser capaz y tener las siguientes facilidades:
    - Ayudar al Grupo de Aseguramiento de Calidad a elaborar el Plan de Inspección.

- Registrar los resultados del Grupo de Aseguramiento de Calidad por cada uno de los proyectos desarrollados.
- Registrar los grupos que llevan a cabo la inspección.
- Ayudar a los programadores, a los usuarios, administradores de proyecto, al equipo de inspección y a los especialistas en una determinada área y fase de desarrollo, a detectar los defectos.
- Ayudar a elaborar los grupos de inspección.
- Registrar las Listas de Comprobación para cada uno de los proyectos y dar alternativas para la creación de nuevas listas.
- Capturar los defectos encontrados durante la ejecución de las inspecciones realizadas en los distintos niveles de desarrollo, para ayudar a prevenir los posibles defectos y mejorar el proceso de inspección.

La hipótesis para el presente trabajo será:

Con las listas de comprobación se puede ayudar a realizar el proceso de Inspección de Software de forma objetiva.

Para cumplir los objetivos y responder a la hipótesis que el autor plantea, se realizaron las siguientes tareas en forma cronológica en tiempo de la siguiente forma:

- Se realizó un estudio del estado actual de las Listas de Comprobación. Para este fin, fue hecha una búsqueda exhaustiva en Internet y en libros relacionados con el desarrollo de software, reflejada en la bibliografía.
- Se estudió los distintos modelos utilizados hoy en día para realizar el proceso de inspección de Software. Para este fin, se realizó una búsqueda y selección de sitios en Internet y lecturas relacionadas con las inspecciones de software.
- Se estudió como participa las inspecciones de software junto con los modelos de: “Modelo de Madurez de Capacidad” [56] y el “Modelo de Madurez de Capacidad Integrado” [62] [63]. Este estudio sirvió para la búsqueda de las Listas de Comprobación adecuadas para el Proceso de Software.

- Se revisó normas ISO para el ciclo de vida del software, estableciendo una nueva clasificación de las listas de comprobación.
- Se estableció un modelo de inspección de software de acuerdo a la situación actual de tiene el país, utilizando el análisis y reflexión de los distintos modelos que se practican en el mundo sobre las inspecciones de software.
- Se ha clasificado las listas de comprobación de acuerdo a los distintos procesos y propiedades que tiene el desarrollo de software.
- Se ha definido una arquitectura de software para la aplicación que ha automatizado el proceso de inspección de software propuesto.
- Se ha logro obtener un conocimiento aceptable sobre la plataforma de desarrollo Visual Studio .Net de Microsoft.
- Se realizó una validación del uso de las listas de comprobación y el modelo de inspección en la Universidad de Ciencias Informáticas y dentro del Centro de Referencia de Ingeniería de Sistemas.

Se realizo un sitio Web, para la preparación de los participantes en el modelo establecido en el presente trabajo sobre las inspecciones de software.

CAPITULO 1 LA CALIDAD Y LAS INSPECCIONES DE SOFTWARE

1.1 INTRODUCCIÓN

En el presente capitulo se presenta una breve introducción a los conceptos, metas y objetivos del aseguramiento de calidad de software, definiendo también, las características que debe tomarse en cuenta para satisfacer los distintos atributos de calidad descritos en el capitulo. Estos atributos, serán rastreados durante el desarrollo proyecto de software para lograr una mejora en el proceso de software.

También, se presenta una introducción a las inspecciones de software, presentando los objetivos que deben lograse al aplicar este proceso. Conjuntamente se presenta, una vista de los distintos modelos mas utilizados internacionalmente para las inspecciones de software. Se explica los objetivos de las listas de comprobación y como ayudan con el proceso de inspección; se aclara, que con las inspecciones de software se logra la detección de los defectos en un artefacto desde el inicio del proyecto de software hasta el final, por este motivo, se presenta una posible clasificación de los defectos que se pueden encontrar, hecha por José Javier Dolado Cosín [10], para culminar se realiza una explicación de las reglas que deben tomarse en cuenta al aplicar las listas de comprobación y como dar una la valoración de una lista.

Para culminar con el capitulo, se presenta un resumen de las métricas de inspección de software y el software existente en el mercado que automatiza el proceso de inspección de software.

Este capitulo dará una idea general de la importancia de las inspecciones de software en cualquier empresa desarrolladora de software.

1.2 ASEGURAMIENTO DE LA CALIDAD DE SOFTWARE (SOFTWARE QUALITY ASSURANCE - SQA)

El aseguramiento de Calidad es una actividad, para toda empresa que desarrolla software y debe llevarla a cabo de una manera limpia y correcta. Es un proceso de la ingeniería de software, que se define como un “conjunto de acciones planificadas y sistemáticas que son necesarias para proporcionar la confianza adecuada en que un producto o servicio cumpla

con los requisitos dados sobre la calidad" [46]. La medición y las métricas son un punto muy importante en todo el proceso de software, como también, el aseguramiento de calidad que se retroalimenta con los resultados de las métricas. La obtención de software con buena calidad implica la utilización de: un modelo de proceso, una metodología de desarrollo, que ayude a cumplir estrictamente el modelo de proceso seleccionado, estos procesos son entre otros: procesos determinación de requisitos, diseño, implementación y prueba, que permiten estandarizar el conocimiento del trabajo, para lograr una calificación adecuada en los "atributos de calidad descritos por McCall" [13], políticas de control para todas las fases de desarrollo y un proceso de medición para el proceso de software.

El Aseguramiento de Calidad de Software engloba los siguientes puntos:

- El mejoramiento de los métodos, técnicas de análisis, diseño, codificación y prueba [6].
- "Revisiones técnicas formales que se aplican durante cada fase del proceso de desarrollo de software" [6], que ayudan a detectar los defectos.
- "Utilización de estándares" [47] durante el desarrollo.
- "Sistema de Métricas" [47], para la retroalimentación de todas las personas
- Definir estrategias de prueba multiescala [6];
- Control de la documentación del software y de los cambios realizados [6];
- Un procedimiento que asegure, siempre que sea posible, un ajuste a los estándares de desarrollo del software. [6]

En el caso de la IEEE 1074, "este estándar explica como el aseguramiento de calidad de software debe apoyarse o relacionarse estrechamente son las siguientes actividades:

- Verificación: Básicamente revisiones y auditorias de configuración y calidad.
- Validación: Todos los niveles y fases de prueba de ejecución de software.
- Gestión de Configuración: Como medio de control de los productos generados.
- Medición de software: Contempla la necesidad de marcar objetivos y asociar métricas a los objetivos. [46]"

La actividad del SQA, en empresas grandes, será llevada por un grupo disciplinado, que deben seguir objetivos y metas, que se describen en los epígrafes siguientes.

1.2.1 OBJETIVOS DEL SQA

Entre los objetivos que persigue el SQA [55], para delimitar su campo de acción y no confundir con otras actividades se definen los siguientes:

- El SQA consiste en la revisión de los productos y su documentación relacionada, para verificar su contenido, corrección, confiabilidad y facilidad de mantenimiento.
- La garantía de que un sistema cumpla las especificaciones y los requisitos funcionales y no funcionales para el desempeño deseado.
- Evitar la necesidad de realizar cambios significativos, una vez que el producto se ha terminado
- Desarrollar software al que sea fácil mejorarlo.
- Garantizar que el proceso se lleve de acuerdo con los estándares establecidos internacionalmente, para ello toma, entre varias herramientas, las Inspecciones, para evaluar, verificar y controlar los proyectos en ejecución, que forma parte de los objetivos de las Revisiones y Técnicas Formales, que “es una actividad del aseguramiento de calidad llevada acabo por los ingenieros del software” [17].

Durante el trabajo, se aclara la función de las Listas de Comprobación y el como contribuyen con los objetivos que persigue el SQA, ya que éstas están incluidas como una herramienta de las Revisiones y Técnicas Formales que contribuyen al cumplimiento de los requisitos.

1.2.2 METAS DEL SQA

El Modelo de Capacidad de Madurez (CMM) y Roger Pressman definen para el SQA las siguientes metas que deben contribuir al mejoramiento del proceso:

- Meta 1: Las actividades de aseguramiento de la calidad del software (SQA) se planifican. [3] [17]
- Meta 2: Se verifica de una manera objetiva la concordancia de los productos de software y de las actividades con los estándares, procedimientos, y requisitos aplicables. [3]
- Meta 3. Se posee la descripción del proceso de software descrito y se actualiza sistemáticamente. [17]
- Meta 4: Se revisan las actividades de ingeniería de software para verificar el ajuste al proceso de software definido. [17]
- Meta 5: Se realizan auditorias de los productos de software designados para verificar el ajuste con los productos definidos como parte del proceso de software. [17]
- Meta 6: Se asegura que las desviaciones del trabajo y los productos del software se documenten y se manejen de acuerdo con un procedimiento establecido. [17]
- Meta 7: Se registra lo que no se ajuste a los requisitos y se informa a sus superiores. [17]

1.2.3 ACTIVIDADES DEL SQA

Según CMM el SQA debe realizar las siguientes actividades [56][58]:

1. Un plan de aseguramiento de la calidad del software es preparado para el proyecto de software siguiendo un procedimiento documentado.
2. Las actividades del Grupo de Aseguramiento de Calidad de Software (GSQA) se realizan de acuerdo al plan de de aseguramiento de calidad.
3. El GSQA participa en la preparación y revisión del plan del proyecto de desarrollo de software, estándares, y procedimientos.
4. El GSQA revisa las actividades de ingeniería de software para verificar su conformidad de cada una de ellas.
5. El GSQA audita los productos del trabajo de software para verificar su conformidad.

6. El GSQA periódicamente reporta el resultado de sus actividades al Grupo de Ingeniería del Software (GIS).
7. Las desviaciones detectadas en las actividades de software y en los productos del trabajo de software son documentadas y manejadas siguiendo un procedimiento documentado.
8. El GSQA conduce revisiones periódicas de sus actividades y hallazgos con el cliente, de la forma más apropiada.

1.3 CALIDAD DE SOFTWARE

El término calidad del software se interpreta de diferentes maneras. Uno de los modelos más difundidos y utilizados de calidad es debido a McCall (1977), "que especifica una serie de atributos, con los cuales es posible tratar de medirla. Sin embargo, existe una visión distinta, es la definición de atributos juntamente con el usuario o asociar la calidad a la ausencia de defectos en el transcurso del desarrollo y vida del software. [10]" Tomar los atributos de calidad de McCall, es comprometerse a realizar un seguimiento durante el transcurso del desarrollo del proyecto de software a estos atributos, además deben estar bien definidos en los requisitos no funcionales y delimitar el limite de cada uno de estos en el diseño de la Arquitectura de Software.

Sin embargo, existen otros dos modelos que tratan, también, de la calidad de software, uno es de Boehm (1978) y de la ISO-9126, con algunas diferencias sobre el modelo de McCall. En la Tabla 1.1 [7] se enumera los distintos atributos y metas de los tres modelos.

Criterio / Meta	McCall, 1977	Boehm, 1978	ISO-9126, 2000
Corrección	X	X	Se incluye en la facilidad de Mantenimiento
Confiabilidad	X	X	X
Integridad	X	X	
Facilidad de Uso	X	X	X

Eficacia	X	X	X
Facilidad de Mantenimiento	X	X	X
Facilidad de prueba	X		Se incluye en la facilidad de Mantenimiento
Interoperabilidad	X		
Flexibilidad	X	X	
Reutilización	X	X	
Portabilidad	X	X	X
Claridad		X	
Fácil de modificar		X	Se incluye en la facilidad de Mantenimiento
Documentación		X	
Elasticidad		X	
Comprensibilidad		X	
Validez		X	Se incluye en la facilidad de Mantenimiento
Funcionalidad			X
Generalidad		X	
Economía.		X	

Tabla 1.1. Modelos de Calidad de Software. "X" significa que se explica el atributo en el modelo.

Dentro de la calidad de software, se considera distintos servicios que puede ofrecer el desarrollador luego de la venta del producto de software. Estos servicios, hacen que los usuarios inclinen su preferencia. Significa, que los servicios adicionales luego de la venta (por ejemplo, el soporte técnico o el mantenimiento del software), deben cumplir una cierta

calidad. La inclinación de los usuarios hacia el producto se debe al cumplimiento de los distintos atributos de la calidad, estándares, requisitos y las características exclusivas del software que se han seguido en el transcurso del desarrollo. Por este motivo, si no se controla la calidad, la empresa de desarrollo no puede conocer si va incrementando gradualmente su madurez en el proceso.

Una definición que ayuda a entender la calidad de software es la siguiente:

> La satisfacción del cliente es la validación final de la calidad. La calidad del proceso, del producto y la satisfacción del cliente, conforman el significado total de la calidad. [16]

Por lo tanto la calidad de software se puede definir de la siguiente manera:

> Es el grado en el que el software satisface una serie de requisitos de operación preestablecidos, los estándares de desarrollo especificados con anterioridad y las características inherentes a todo producto de software desarrollado de manera profesional. [5]

Existen tres puntos importantes respecto a la definición de la calidad de software que menciona Pressman [17] y son:

1. Los requisitos del software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.
2. Los estándares definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería de software. Si no se siguen esos criterios, casi siempre habrá falta de calidad.
3. Existe un conjunto de requisitos implícitos que a menudo no se mencionan. Si el software se ajusta a sus requisitos explícitos pero falla en alcanzar los requisitos implícitos, la calidad del software queda en entredicho.

De este modo, para que la calidad llegue a cumplirse se debe realizar un control hacia esta y se puede definir de la siguiente manera:

> El Control de Calidad es el conjunto de técnicas y actividades de carácter operativo, utilizadas para verificar los requisitos relativos a la calidad del producto y del servicio. [5]

Obtener buena calidad en el desarrollo del software es un reto mas difícil de enfrentar que en otras actividades creativas e industriales. “Existen metodologías y mecanismos para establecer programas que conducen directamente a que cada uno de los involucrados hagan las cosas cada vez mejor.” [4]

Para que tengan validez estas definiciones a la hora de entregar un producto de software se toma muy en cuenta los atributos que señala McCall (modelo más utilizado) u otros modelos, que serán las guías para el desarrollador. A continuación, se explica brevemente cada uno de los atributos.

1.3.1 ATRIBUTOS DE CALIDAD SEGÚN MCCALL

En definitiva “el modelo de MaCall es una fuente de ideas para evaluar la calidad, pero hay que recordar que no esta validado ni en su estructura de descomposición, ni en la manera de combinar valores de nivel para evaluar otro superior, ni en la adecuación de las métricas propuestas para cada criterio. [46]”

1.3.1.1 PARA LA OPERACIÓN

Capacidades Operativas.

Corrección.

¿Hace lo que se le pide?

Grado en el que un programa satisface las especificaciones y cumple los objetivos del usuario. [5] [6] [7] [10] [13] [18] [46]

- Completitud: Atributo del software que proporcionan la implementación completa de todas las funciones requeridas.
- Consistencia: Atributo de software que proporcionan uniformidad en las técnicas y notaciones de diseño e implementación.
- Fácil de Rastrear: Atributo del software que proporciona la posibilidad de una traza desde los requisitos a la implementación en un entorno operativo correcto.

Un programa debe operar correctamente o proporcionará poco valor a sus usuarios. La corrección es el grado en el que el software lleva a cabo la función requerida. La medida más común de corrección son los defectos por KLDC[1], en donde un defecto se define como una falta verificada de conformidad con los requisitos.

Fiabilidad o Confiabilidad

¿Lo hace de forma fiable todo el tiempo?

La probabilidad de que un programa realice su objetivo satisfactoriamente (sin fallos) en un determinado periodo de tiempo y en un entorno concreto (denominado perfil operacional). [6] [7] [10] [13] [46]

- Precisión: Atributo del software que proporcionan el grado de precisión requerido en los cálculos y resultados.
- Consistencia: Atributo de software que proporcionan uniformidad en las técnicas y notaciones de diseño e implementación.
- Modularidad: Atributo del software que proporciona una estructura de módulo altamente independiente.
- Simplicidad: Atributo del software que posibilitan la implementación de funciones de la forma más comprensible posible.
- Exactitud: Atributo del software que establece la precisión de los cálculos y del control de los datos.
- Madurez. Capacidad de que el software evite una falla. [42]
- Tolerancia a fallos: Atributo del software que posibilita la continuidad del funcionamiento bajo condiciones no usuales. [42]

[1] Mil líneas de código

- Facilidad de recuperase. La capacidad del software para restablecer un nivel de ejecución y recuperar datos directamente afectados en el caso de una falla. [42]
- Conformidad. La capacidad del software para adherirse a estándares, convenciones o regulaciones relacionadas a la confiabilidad. [42]

Eficiencia

¿Qué recursos hardware y software se necesitan?

Cantidad de recursos y código requeridos por un programa para realizar una función o ejecutar un proceso establecido. [5] [6] [7] [10] [13] [46]

- Eficiencia en ejecución. Atributo del software que minimizan el tiempo de realización de los procedimientos.[42]
- Eficiencia en almacenamiento. Atributo del software que minimizan el espacio de almacenamiento necesario.
- Utilización de recursos. La capacidad del software para el uso apropiado del tiempo con el desempeño de funciones antes establecidas.

Integridad

¿Se puede controlar su uso?

Grado en el que se controla el acceso al programa o los datos por usuarios no autorizados. [5] [6] [7] [10] [13] [46]

- Control de accesos: Atributo del software que proporcionan el control de acceso al software y los datos que maneja.
- Facilidad de auditoria: Atributo del software que facilitan la auditoria de lo accesos al software.
- Seguridad: La disponibilidad de mecanismos que controlen o protejan los programas o los datos.
- Amenaza: La probabilidad (que se puede estimar o deducir de la evidencia empírica) de que un ataque de un tipo determinado ocurra en un tiempo determinado. [18]

La integridad puede violarse por ataques que se puede realizar en cualquiera de los tres componentes del software: programas, datos y documentos

Facilidad de Uso

¿Es fácil y cómodo de manejar?

Esfuerzo necesario para aprender, operar, preparar entradas e interpretar la salida de un programa. [5] [6] [7] [10] [11] [13] [46]

- Facilidad de Operación: Atributo del software que determinan la facilidad de operación del software. [42]
- Facilidad de Comunicación: Atributo del software que proporcionan interfaz de entrada y salida fáciles de utilizar y amigables.
- Facilidad de aprendizaje: Atributo del software que facilita la familiarización inicial del usuario con el software y la transición del modo actual de operación. [42]
- Formación: El grado en el que el software ayuda para permitir que nuevos usuarios apliquen el sistema.
- Entendimiento. La capacidad del producto de software para permitir al usuario entender si el software es adecuado, y como puede ser usado para tareas particulares y variadas condiciones de uso. [42]
- Atractivo. La capacidad del producto de software para ser gustado por el usuario.
- Conformidad. La capacidad del producto de software para adherirse a los estándares, convenciones, guías de estilo o regulaciones relativas a la facilidad de uso. [42]

1.3.1.2 PARA SU REVISIÓN

Capacidad para soportar cambios.

Facilidad de Mantenimiento

¿Se puede localizar fácilmente los fallos?

Esfuerzo requerido para localizar y corregir un error en un programa en funcionamiento. [5] [6] [7] [10] [11] [13] [46]

- Modularidad: Atributo del software que proporciona una estructura de módulos altamente independiente.
- Simplicidad: Atributo del software que posibilitan la implementación de funciones de la forma más comprensible posible
- Consistencia: Atributo de software que proporcionan uniformidad en las técnicas y notaciones de diseño e implementación
- Concisión: Atributo del Software que posibilitan la implementación de una función con la menor cantidad de código posible.

- Auto descripción: Atributo del software que proporcionan explicaciones sobre la implementación de las funciones.
- Facilidad de análisis. La capacidad del producto de software puede ser diagnosticado por deficiencias o causas de falla o partes para ser modificadas. [42]
- Facilidad de cambio. La capacidad del producto de software para permitir una modificación especificada para ser implementada. [42]
- Estabilidad. La capacidad del producto de software para minimizar efectos inesperados de las acciones que están codificadas en el software. [42]
- Fácil de Comprobarse. La capacidad del producto de software para permitir ser validado. [42]
- Conformidad. Capacidad del producto de software para adherirse a estándares o convenciones relativas a la facilidad de mantenimiento. [42]

Existen cuatro tipos de mantenimiento:

- Mantenimiento correctivo: Corregir los errores.
- Mantenimiento adaptativo: Modificar el software de acuerdo con el entorno.
- Mantenimiento perfectivo: Modificar el software de acuerdo a los cambios que se efectúan en los procesos del sistema a medida que pasa el tiempo.
- Mantenimiento preventivo: no está tan extendido y consiste en cambiar el producto pensando en mejoras futuras.

No hay forma de medir directamente la facilidad de mantenimiento; por consiguiente, se deben utilizar medidas indirectas. Una simple métrica orientada al tiempo es el tiempo medio de cambio (TMC), es decir, el tiempo que se tarda en analizar la petición de cambio, en diseñar una modificación adecuada, en implementar el cambio, en probarlo y en distribuir el cambio a todos los usuarios. Como media, los programas que son más fáciles de mantener tendrán un TMC más bajo (para tipos equivalentes de cambios) que los programas que son más difíciles de mantener. [18]

Facilidad de prueba

¿Se pudieron probar todas las opciones?

Esfuerzo requerido para probar un programa que garantice la función deseada. [5] [6] [7] [10] [13] [46].

- Modularidad: Atributos del software que proporciona una estructura de módulo altamente independiente.
- Simplicidad: Atributos del software que posibilitan la implementación de funciones de la forma más comprensible posible
- Auto descripción. Atributo de software que proporciona una descripción por si solo, es decir, que el software debe describirse por si solo.
- Instrumentación: Atributo del software que posibilitan la observación del comportamiento del software durante su ejecución para facilitar las mediciones del uso o la identificación de errores.

Flexibilidad

¿Se puede añadir nuevas opciones?

Esfuerzo requerido para modificar un programa en funcionamiento. [5] [6] [7] [10] [13] [46].

- Auto descripción. Atributo del software que permita la auto especificación.
- Capacidad de expansión: atributo el software que posibilitan la expansión del software en cuanto a capacidades funcionales y datos.
- Generalidad: Atributo del software que proporcionan amplitud de la funciones implementadas.
- Modularidad: Atributos del software que proporciona una estructura de módulo altamente independiente.

1.3.1.3 PARA SU TRANSICIÓN

Adaptabilidad a nuevos entornos.

Portabilidad o Facilidad de Transportación

¿Se puede usar en otras máquinas?

Esfuerzo requerido para transferir un programa de una configuración de hardware o entorno de software a otro. [5] [6] [7] [10] [11] [13] [46].

- Auto descripción. Atributo del software que permita la auto especificación.
- Modularidad: Atributo del software que proporcionan una estructura de módulo altamente independiente.

- Independencia entre sistema y software. Atributo del software que describe la capacidad de ejecución en distintas plataformas.
- Independencia del hardware. Atributo del software que describe la capacidad de ejecución en distintas configuraciones de hardware.
- Instalación. La capacidad del software de ser instalado en un específico ambiente. [42]
- Coexistencia. La capacidad de coexistir con otro software en un ambiente común compartiendo recursos comunes. [42]

Capacidad de Reutilización

¿Se puede usar alguna parte del software en otra aplicación?

Grado en el que un programa se puede utilizar en otras aplicaciones. [5] [6] [7] [13] [46].

- Auto descripción. Atributo del software que permita la auto especificación.
- Modularidad: Atributo del software que proporcionan una estructura de módulo altamente independiente.
- Independencia entre sistema y software: Atributo del Software que determinan su dependencia del entorno operativo.
- Independencia de hardware: Atributo del software que determinan su dependencia del hardware.

Interoperabilidad

¿Se puede comunicar con otras aplicaciones o sistemas informáticos?

Esfuerzo requerido par acoplar un sistema con otro. [5] [6] [7] [10] [13] [46].

- Modularidad: Atributo del software que proporcionan una estructura de módulo altamente independiente.
- Compatibilidad de comunicaciones: Atributo del software que posibilita el uso de protocolos de comunicación e interfaces estándar.
- Compatibilidad de dato: Atributo del software que posibilita el uso de datos en representaciones estándar.
- Estandarización en los datos: El uso de estructuras de datos y de tipos estándares a lo largo de todo el programa.

1.4 INSPECCIONES DE SOFTWARE

Muchas de las pirámides de Egipto construidas alrededor de los años 2000 y 3000 años A. C. tienen parámetros que las acercan casi a la perfección en la construcción pues en la orientación de la base de alineación N-S, E-O el error máximo llega a ser de 6 minutos de arco, distando la base de algunas de ellas de ser un cuadrado perfecto en menos de 17,78 cm. Todo ello se conseguía gracias a los métodos de inspección empleados durante su construcción [13].

Las inspecciones de software fueron definidas por Fagan, al principio de los años 70 para la IBM, no eran más que exámenes estrictos dirigidos al código fuente. En la actualidad, esta dirigido a los procesos, metodologías, planes o a todo el ciclo de vida, es decir, a cualquier artefacto producido en el transcurso del desarrollo; detectando defectos en estos.

Las inspecciones son parte fundamental del aseguramiento de calidad, establecen un orden en el proceso y garantizan la mejora continua del proceso. Es un proceso de mejora de calidad continuo. “Trata el producto, pero también el proceso de desarrollo así como su propio proceso. [9]”

Por ejemplo, dentro del ciclo de vida, las inspecciones se realizan al final de cada una de las etapas como se ve en la figura 1.1. Por otra parte, también se realiza inspecciones a la documentación que especifica la planificación de las distintas etapas de desarrollo; como se ve, también, en la figura 1.1.

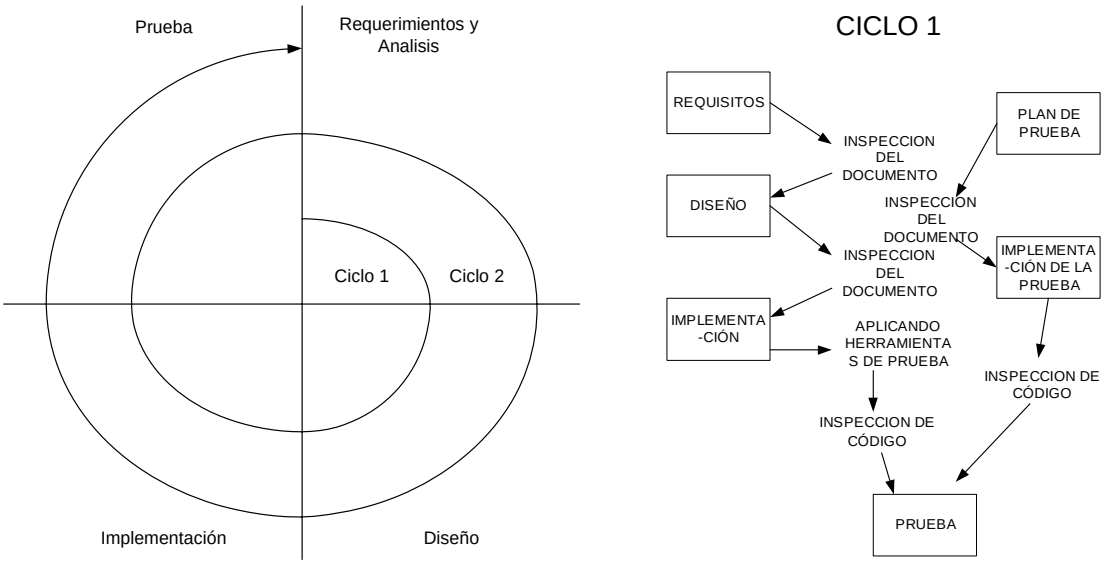


Figura 1.1. Inspecciones en el Ciclo de Vida.

Se puede detectar defectos antes de continuar a una nueva etapa de desarrollo, sea en el ciclo de vida o en los distintos planes que aseguran el desarrollo. Para continuar el diseño. Primero, se realiza una inspección a los requisitos obtenidos. Se comprueba que los requisitos posean una buena documentación, así mismo, cuando se continua con la implementación se realiza una inspección que compruebe la buena especificación del diseño del nuevo sistema y se continua hasta la prueba del software.

De esta misma forma, se inspecciona los distintos planes que llevarán al éxito y culminación del desarrollo del software, planes como el Plan de Configuración de Software, Plan de Aseguramiento de Calidad, Plan de la Determinación de Requisitos, Plan de Prueba, etc. De este modo, las inspecciones se convierten en una herramienta muy útil para la mejora continua de la calidad de software detectando tempranamente los defectos.

Las inspecciones son convenientes para los proyectos, las organizaciones de soporte y es un apoyo para varias Áreas de Proceso dentro del modelo de CMM o CMMI. [14]

Como se dijo anteriormente, las inspecciones son un proceso de mejora incremental, es decir que a medida que pasa el tiempo en el desarrollo de software los defectos deben reducirse.

Pero, ¿por qué es un proceso de mejora incremental en el desarrollo de software? La respuesta esta en la Figura 1.2, tomada de la University of Oulu, Department of Infomation Processing Science [19].

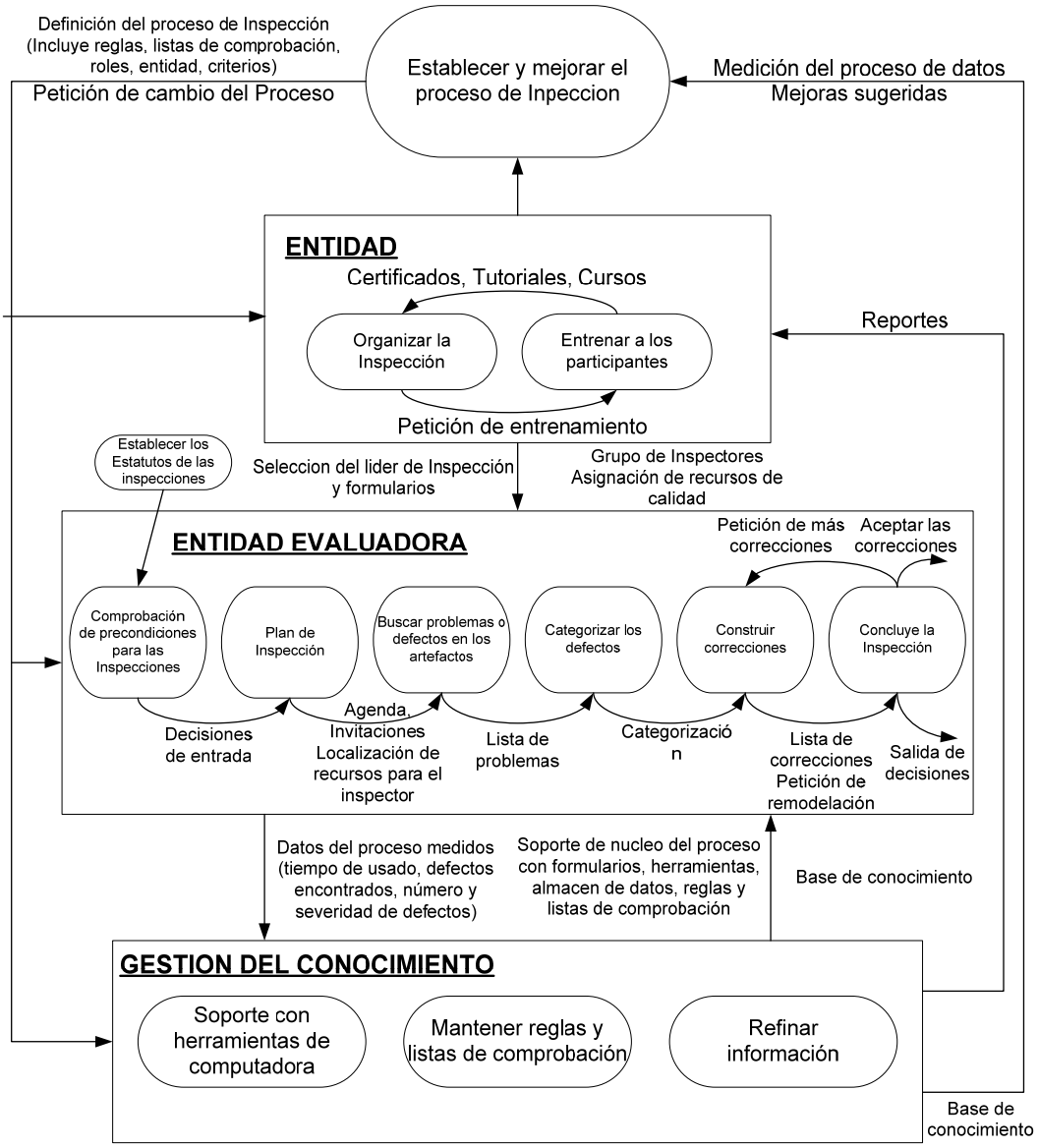


Figura 1.2. La puesta en práctica de la Inspecciones.

Hay seis metas definidas que conforman la práctica de las inspecciones: (1) para identificar defectos, (2) para estimar la calidad, (3) para mejorar la calidad del producto, (4) para proporcionar los datos para la mejora de proceso (Métricas), (5) para proporcionar los

medios para la transferencia del conocimiento, y (6) para mejorar la eficacia del proceso del desarrollo. La práctica de las inspecciones se clasifica en tres sistemas (Figura 1.2.): actividades de soporte (Gestión del Conocimiento), que ayudan a realizar el continuo proceso de inspección, el sistema de base de las actividades de evaluación (Entidad Evaluadora) que son la esencia de la puesta en práctica del proceso de la inspección, y las actividades de la organización (Entidad), que aseguran la mejora y organización eficiente del proceso de la inspección. Las actividades de soporte son el "Soporte con herramientas de software", "Mantener reglas y las listas de comprobación" y "Refinar la información". El sistema actividades, incluye "las condiciones previas para la inspección", "plan de la inspección", "buscar los problemas y defectos en los artefactos", "categorizar los defectos", "construir las correcciones" y "concluir la inspección", mientras que la revisión el sistema consiste en el resto de las actividades, por ejemplo "organizar la inspección", "entrenan a los participantes" y "establecer y mejorar el proceso de la inspección".

Los fabricantes del software entienden que el desarrollo del software es un proceso de experimentación que involucra el descubrimiento continuo de información técnica, asociada con las funciones, formularios y actividades internas que aseguran el producto.

Las Inspecciones del software son una actividad del razonamiento realizada por desarrolladores que juegan los papeles definidos de moderador, registrador, crítico, y lector. Cada papel lleva con él las conductas específicas, habilidades, y conocimiento que necesitan para lograr la práctica de especialistas de Inspecciones del Software.

El ahorro obtenido con la realización de las inspecciones de software se deriva del temprano descubrimiento y correspondiente corrección de defectos que evitan el aumento del costo en el proceso de software. Un defecto no detectado, que escapa a la próxima fase, puede costar de dos a diez veces más para descubrirlo y corregirlo [17].

1.4.1 OBJETIVO DE LAS INSPECCIONES

Las inspecciones deben ser muy bien planificadas y bien llevadas, no puede olvidarse ningún detalle y siempre se debe tener en mente los siguientes objetivos:

- Encontrar tempranamente los defectos.
- Prevenir el mal funcionamiento de los procesos o planes establecidos.
- Proporcionar mejoras en la fiabilidad, disponibilidad, y la facilidad de mantenimiento del software.
- Descubrir continuamente la información técnica, asociada con las funciones, formularios y actividades internas que aseguran el producto.
- Continuar el mejoramiento del proceso de desarrollo.
- Establecer una igualdad de conocimiento dentro de los desarrolladores para la buena practica de los estándares y técnicas de desarrollo.

La ventaja más reconocida para las inspecciones, es la integración y la educación implícita de las personas que sean nuevas en el proyecto, ganando mucho tiempo en la preparación y capacitación.

1.4.2 MÉTRICAS EN EL PROCESO DE INSPECCIÓN DE SOFTWARE

El área de las métricas de software, es una de las áreas en la ingeniería de software donde se ha investigado desde los años de los 70s, ayudando, las métricas, a justificar la calidad del software y a contribuir con el mejoramiento de la calidad.

Todas las ramas de la ciencia se basan en un sistema de métricas que permiten establecer un nivel de conocimiento, estableciendo métodos de medición o también conocida como “La teoría de medición” que permiten lograr que la descripción de las métricas aporten al sistema de una forma objetiva, exacta, reproducible, segura y significativa [24]. Es muy importante que la medición sea estricta y de cobertura amplia y necesaria al sistema.

Además, las métricas son muy importantes en cualquier proceso de desarrollo. Ellas serán las que den parámetros para medir si el proceso está correctamente implantado y se lleva de acuerdo a los objetivos de la empresa.

Existe una diferencia entre medida, medición y métrica, ya que son tres conceptos muy diferentes. Para entender mejor la diferencia se da la definición de los tres conceptos:

- Medida. Proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto. Pueden ser directas, por ejemplo, número de líneas de código, número de errores encontrados, etc., o pueden ser indirectas, por ejemplo, funcionalidad, calidad, complejidad, etc.
- Medición. Acto de determinar una medida.
- Métrica. Es una medida cuantitativa del grado en que un sistema o proceso posee un atributo dado. Por lo general relaciona una o más medidas, por ejemplo, número de errores encontrados por cada mil líneas de código.

Pero en el campo del software un sistema de métricas permite la continua aplicación de técnicas basadas en la medición al proceso de desarrollo de software y a sus productos para proveer información administrativa significativa y oportuna, junto con el uso de esas técnicas para mejorar el proceso y sus productos. [15]

Las empresas fabricantes de software definen sus propias métricas para distintas actividades, que le ayudan a establecer una mejora de su proceso de desarrollo, es claro que las métricas son muy diversas para un propósito, que va de lo general a lo particular, como se ve en [50].

A veces, “la calidad se relaciona con medidas de tiempo o de esfuerzo (por ejemplo, defectos / Hora) y se puede denominar Tasa de Defecto. Este tipo de medida se puede emplear para evaluar la calidad del propio proceso de detección de defectos (por ejemplo, de las pruebas de software) más que para evaluar la calidad del producto de software. En otros casos se sugiere relacionar los defectos con el tiempo necesario para corregirlos, como indicador del proceso de mantenimiento de software.

Para el presente trabajo se selecciona de Rico (1996) [64], seis clases de métricas y treinta y nueve métricas individuales para lo que él llamó “Métrica del Proceso de Inspección del Software" para ayudar al mejoramiento del proceso del software, como se ve en la tabla 1.2.

Clase Métrica	Métrica	Modelo
Planeamiento.	Rico	$SLOC^2$/(Tasa* 2) * (Tamaño del Equipo * 4 + 1)
	Hewlett Packard	SLOC/(Tasa * 2) * 25
	AT&T	50 * KSLOC
	Investigación Norteña De Bell	3 * KSLOC * 4 * 8
	Tom Gilb	SLOC/(Tasa * 2) * (5,76 * Tamaño Del Equipo)

Clase Métrica	Métrica
Total	Defectos por hora de Inspección. Defectos Importantes. Defectos importantes por hora de Inspección. Defectos de menor importancia. Defectos de menor importancia por hora de Inspección. Eficacia del Retiro del Defecto. Horas Totales. Duración. Personas.
Tasa[3] de la Revisión	Tasa de la Descripción. Tasa de la Preparación. Tasa de la Inspección. Tasa de la Reanudación.
Duración de Actividades Internas	Horas de Planeamiento. Horas de Descripción. Horas de Preparación.

[2] SLOC: Número de líneas fuente, excepto espacios en blanco y comentarios
[3] Tasa: Medida sobre el tiempo: Por ejemplo: Defectos por hora.

	Horas de Inspección. Horas de Reanudación. Horas de Continuidad en la inspección.
Intervalo entre las Actividades Internas.	Intervalo entre la Planificación y Descripción. Intervalo entre Descripción y Preparación. Intervalo entre Preparación e Inspección. Intervalo entre Inspección y Corrección. Intervalo entre la Corrección y el Seguimiento. Intervalo entre la Planificación y la Preparación. Intervalo entre la Planificación y la Inspección. Intervalo entre Planificación y Corrección. Intervalo entre Planificación y Seguimiento. Intervalo entre Inspección y Seguimiento
Eficacia de las Actividades Internas	Eficacia de la Preparación. Eficacia de la Inspección. Tasa del Tiempo de aumento en la Inspección. Cancelar la Inspección.

Tabla 1.2: Métricas del proceso de la inspección de software para la mejora de proceso de software (SPI)

A pesar de que pudiera “parecer que el número de defectos del software es un buen indicador de su calidad, en realidad lo que estamos midiendo es la ausencia de la calidad. Un defecto es cualquier elemento erróneo en el software que implica no cumplir los requisitos fijados. Lo cierto, es que en la práctica muchas veces se reduce el concepto de defecto a las causas de fallos funcionales, lo que, en buena medida, se debe a que los requisitos no funcionales no suelen estar bien definidos. En cuanto a la relación entre defectos y fallas del software, en Adans (1984) se puede encontrar que un reducido número de defectos (alrededor del 2%) puede provocar una gran cantidad de fallas, así como hay sistemas con bastantes defectos que fallan pocas veces. [46]”

1.4.3 MODELOS ACTUALES DE LAS INSPECCIONES

Existen muchos modelos metodológicos para realizar inspecciones de software. Todos tienen la meta de detectar los defectos. Este proceso fue iniciado por Fagan en 1976 para la empresa IBM, realizando una copia del proceso de inspección de los componentes hardware, para luego adecuar al código fuente del software. Está basado en un grupo de personas que actúan como examinadores llamados inspectores, los cuales deben tener los siguientes conocimientos y niveles de preparación:

- Debe existir ingenieros de hardware o software, que han estado involucrados en el desarrollo del producto. Conocen bien el producto y están capacitados para generar soluciones de diseño. Los diseñadores de alto nivel aportan un valor adicional en la inspección. [23]
- Un ingeniero de un producto ya existente o relacionado, con aquel sometido a estudio sirve como reconocido inspector, no estando influenciado por el conocimiento de la evaluación del producto. Este inspector reconocerá inconsistencias del producto frente a otros similares. [23]
- Debe existir una persona que ha escrito o escribirá la documentación asociada al producto. [23]
- Debe existir una persona con amplia experiencia en la interacción con los usuarios, en la detección de los defectos y en la generación de soluciones. [23]
- Debe existir un ingeniero de mantenimiento que tenga una amplia experiencia con productos similares, así como en la resolución de problemas en el campo. [23]

A medida que el tiempo ha pasado otros investigadores contribuyeron con distintas metodologías para realizar las inspecciones de software siempre con la ayuda de los inspectores, variando el modelo de Fagan, como se muestra continuación.

Modelo de Fagan (1976) [20]

El proceso de la inspección de Fagan implica la revisión sistemática de código o de artefactos relacionados, tales como requisitos y documentos de diseño. La inspección ha

sido acertada, probada en varios usos industriales y está muy difundida en la industria del Software, por estos motivos las inspecciones de Fagan han marcado un camino hacia el mejoramiento del proceso de software.

Un equipo de inspección de Fagan consiste en un asesor, un lector, un inspector, y un autor. El asesor desempeña un papel muy importante de la dirección y debe asegurarse de que la preparación del equipo se centre en la detección de defectos sin desviarse (por ejemplo, sugiriendo correcciones necesarias o resaltar detalles importantes). El papel del lector es parafrasear (explicar y leer) el producto que es inspeccionado a un paso razonable. Si no, el equipo puede inspeccionar muy rápido y superficialmente lo inspeccionado. La presencia del autor en las reuniones de la inspección, generalmente se considera beneficiosa, porque (1) el autor puede asistir al equipo de la inspección para entender mejor el producto, y (2) el autor está preparado para entender la naturaleza exacta de los defectos en los hallazgos del equipo. El papel de un inspector es examinar el software desde el punto de vista de un usuario. El inspector es el responsable de velar por los intereses del usuario. Un equipo que realice una inspección es más productivo cuando sus miembros trabajan en armonía y satisfacen los papeles asignados.

El proceso de la inspección de Fagan consta de los pasos siguientes, cada uno con objetivos específicos: planeamiento, descripción, preparación, inspección, reanudación, y seguimiento.

- **Planificación:** Cuando los materiales para ser inspeccionados pasan por los criterios de entrada (por ejemplo, el código fuente compila con éxito sin errores de sintaxis), miembros del equipo de inspección se seleccionan, y se establecen los horario de la inspección (por ejemplo, tiempo y lugar).
- **Descripción:** Se dan instrucciones previas a los miembros del equipo del material a ser inspeccionado, y se asignan los papeles.
- **Preparación:** Los miembros del equipo estudian el material individualmente para prepararse para satisfacer los papeles asignados.
- **Inspección:** El equipo realiza una reunión de inspección para encontrar defectos, y registrarlos. El propósito de la reunión de la inspección es la detección de los defectos o de violaciones de estándares, y cualquier tentativa de encontrar soluciones

alternativas debe ser eliminada por el asesor.

- **Remodelar:** El autor revisa el resumen de los defectos detectados, clarificando cuales son realmente defectos y que son mal entendidos en el proceso de la inspección. Entonces, el autor debe modificar para corregir los defectos.
- **Seguimiento:** El asesor o el equipo entero de inspección repasa el producto otra vez, para asegurar que todos los arreglos son eficaces y de que no se ha introducido ningún defecto adicional durante la remodelación.

Muchas variaciones se han propuesto sobre el método de inspección de Fagan. Sin embargo, se elige este método en el mundo porque es el más aplicable en la industria del software.

Modelo de Tom Gilb (1993) [21] [22]

Uno de los textos más comprensivos en inspecciones del software es el de Gilb y de Graham. El modelo que describen se basa, obviamente, en el trabajo de Fagan; sin embargo, también incorpora otros pasos. "Un paso adicional es el proceso de la prevención del defecto explicada por Jones" [46].

Hay tres papeles definidos en este tipo de inspección. El líder (Jefe de Aseguramiento de Calidad) está en el puesto principal del proceso y es el que realiza el planeamiento y garantiza el funcionamiento de la inspección. El autor del documento es un participante requerido en la reunión de registro y debe ser parte de la verificación. Los miembros restantes del equipo son los inspectores, que su deber es simplemente encontrar y divulgar defectos en el documento o en el artefacto. Durante la reunión de registro, se asigna el papel de escribano a uno de los inspectores y registra los defectos encontrados durante la inspección.

El modelo de inspección de Gilb consiste en los siguientes pasos:

- **Planeamiento y documentos de entrada.** El líder comienza con asegurarse que los criterios de inicialización están satisfechos. Esto asegura que la inspección esté con los documentos fundamentales y no se pierda ningún detalle. Es seguido por el planeamiento de la inspección, donde el líder determina a los participantes de la inspección y designa a 3-4 como inspectores. Él elabora las listas necesarias de la

documentación, las reglas, los estándares y programa las reuniones. Esta fase produce un plan maestro para la inspección entera.

- **Reunión rápida**. El Jefe de Aseguramiento de Calidad organiza una rápida reunión de 15 minutos, donde él da una escala de tiempo de realización para la inspección y otras instrucciones a los inspectores y al autor, explica en términos generales la estructura de la documentación y el propósito de la inspección.
- **Inspección o comprobación**. Es realizada por cada par individualmente, que registra cada defecto en una tabla.
- **Registro.** Cuando los inspectores han acabado la comprobación en la fecha convenida el líder de la inspección organiza la reunión de registro (máximo 2 horas) donde se mencionan todos los defectos y su aceptación o rechazo en el registro general de la inspección.
- **Tormenta de ideas.** Una reunión de tormenta de ideas (5-30 minutos) sigue poco después de la reunión de registro. Donde se trata de dar solución o ideas a los defectos encontrados para su remodelación.
- **Edición.** Se espera que el autor emprenda la edición del análisis y la acción de corrección.
- **Seguimiento.** El líder de la inspección realiza un seguimiento a los cambios que debe realizar el autor manteniendo un contacto con este.
- **Salida.** Se entrega el producto y está listo para la salida de la inspección cuando todos los puntos discutidos en la tormenta de ideas y el registro se han corregido y trabajado satisfactoriamente.

De la descripción anterior se puede ver la diferencia entre este proceso de Tom Gilb y el de Fagan, es la etapa de la detección de los defectos, es decir durante la fase individual por cada inspector.

Modelo de Humphrey (1989) [21]

El proceso de la inspección descrito por Humphrey es muy similar al descrito por Fagan; sin embargo, hay algunas diferencias importantes. El equipo de la inspección consiste en un número de personas con papeles previstos como: asesor, productor y revisor. Las fases

descritas son virtualmente idénticas en nombre a las descritas por Fagan, pero el proceso real es diferente.

- **Planificación.** La etapa de planeamiento permite la selección de participantes y de la preparación de los criterios de la entrada.
- **Descripción.** La etapa de la descripción es idéntica a la de Fagan.
- **Preparación**. Los inspectores detectan los errores y registran en una sola lista de registro.
- **Análisis**. La lista de defectos o de registro se pasa al autor para que la analice y verifique los defectos y se prepare para la inspección.
- **Inspección**. En esta fase, el desarrollador explica los defectos encontrados y los inspectores aclaran también, el por que de los defectos. De esta forma se establece una lista de defectos que debe pasar a la siguiente fase.
- **Remodelar.** Esta etapa es similar a la de Fagan.
- **Seguimiento.** Esta etapa es similar a la de Fagan.

La diferencia entre el modelo de Fagan y el de Humphrey es en la etapa de Análisis y de Inspección, ya que, deja participar al autor o desarrollador en la detección de defectos.

1.4.4 MÉTODOS DE LA INSPECCIONES

De acuerdo a las características de software, dentro del modelo de inspección de software seleccionado se planifica un método que varían por su funcionalidad, por las personas requeridas para llevar acabo la actividad, por la complejidad del software, por el artefacto y por la experiencia de la institución que desarrolla el software. Existen métodos de inspección específicos para una etapa de desarrollo o para un atributo de calidad. Cada uno de los métodos, que se explican brevemente a continuación, son una herramienta para la detección de los defectos en las inspecciones de software. Sin embargo, el método más utilizado dentro de los modelos de inspección es el que se basa en Las Listas de Comprobación.

- Inspecciones de diseño de alto nivel. Durante la fase de diseño de alto nivel, todo el diseño para un modulo o función es producido. Cada fase comienza con la distribución de los requisitos del software relevantes o especificaciones del plan de

prueba, las especificaciones de la interfaz y finaliza con la realización de la inspección. En esta fase la arquitectura del software es determinada y escrita en la especificación del diseño de software inicial. Esta información de diseño de alto nivel es examinada durante la inspección. Para cada función, la especificación es facilitada. [53]

- Inspecciones de diseño de bajo nivel. Es conocido como diseño detallado o diseño de modulo. Refleja lo insatisfactorio en los objetivos del diseño. Los objetivos importantes considerados en el diseño del software. Verifica que el refinamiento del diseño esté listo para pasar a la codificación. [53]
- Inspecciones de código. Está dirigida fundamentalmente a encontrar defectos en la codificación de las tareas y funcionalidades del producto de software
- Inspecciones formales de Facilidad de Uso. Se toma la metodología de inspecciones de software y se adapta a la evaluación de Facilidad de Uso. Esta técnica también proporciona medidas cuantitativas que pueden ser seguidas mediante métodos estadísticos para el control del proceso. [8]
- Inspecciones de características. Las inspecciones de características analizan únicamente un conjunto de características determinadas del producto, proporcionando escenarios de usuarios para obtener el resultado final del uso del producto. Por ejemplo, un escenario de usuario habitual para el uso de un procesador de textos es escribir una carta. [8]
- Inspecciones de consistencia. El objetivo de la inspecciones de consistencia, es asegurar que los objetivos descritos para el artefacto sean inspeccionados con los múltiples productos desarrollados procedentes del mismo esfuerzo de desarrollo. [8]
- Inspecciones de estándares. Las inspecciones de estándares garantizan el ajuste a los estándares industriales o internacionales. En tales inspecciones, un profesional de la Facilidad de Uso con extenso conocimiento de los estándares, analiza los elementos del producto de acuerdo a su uso y acondicionamiento al estándar industrial.
- Evaluación Heurística. Es una variedad de las inspecciones donde los especialistas juzgan si cada elemento de la interfaz de usuario sigue los principios establecidos. La Evaluación Heurística puede ser utilizada en, prácticamente, cualquier momento del ciclo de desarrollo, aunque probablemente sirva para efectuar una prueba. No

requiere gran instalación de medios y puede ser llevada acabo por personas especializadas o incluso por usuarios seleccionados. [8]

- Paseos Cognitivos (Walkthroughs). Los Paseos Cognitivos derivan de los análisis cognitivos y reciben este nombre porque el especialista que realiza la sesión recorre un escenario de tareas determinadas como lo habría de hacerlo un usuario seleccionado. Según se decida o sea conveniente la sesión será de forma individual o en grupo (Usuarios, desarrolladores o profesionales). [8]
- Inspecciones con Listas de Comprobación. Es un grupo seleccionado de preguntas de forma cerrada y ordenadas por las características, atributos del software o de calidad, o por la experiencia.
    - Guías de Comprobación. Las Guías de comprobación ayudan a asegurar que determinados principios sean considerados en un diseño. Normalmente, se utilizan en combinación con otro método de inspección y sirven de referencia. [8]
    - Basadas en Escenarios. Se puede entender como una particularización de las anteriores en la que la inspección se lleva a cabo a través de tres escenarios: Usuario novato, usuario experto y manejo de errores. Para cada uno se proporciona una lista de aspectos a comprobar. [8]
- Inspecciones con Evaluación Cooperativa.
    - Método de Diario. Se refiere a los usuarios que registren las actividades que desarrollan en su entorno de trabajo durante un día normal. El registro puede tener o no carácter estructurado. [8]
    - Modelo por Empatía. Se trata de un método desarrollado para aplicar con usuarios con discapacidades, de modo que el diseñador o desarrollador trata de ponerse en situación del usuario simulando tal discapacidad. Tal circunstancia es muy compleja, requiriendo amplios estudios e investigaciones. [8]

Los tipos de Inspecciones pueden combinarse, dependiendo de las características del proyecto a revisar. [53]

1.5 OBJETIVOS DE LAS LISTAS DE COMPROBACIÓN

Los objetivos de las listas de comprobación son muy diversos y muy ricos para la mejora del proceso de inspección y para el proceso en general de software. Debe asegurar que los principios y estándares sean considerados en todo el desarrollo del software (ciclo de vida e hitos del proyecto), también debe ayudar a prevenir, descubrir y corregir los defectos en los artefactos para prever su buen funcionamiento y que permita la revisión conveniente, barata y pertinente de las prácticas de la tecnología y la lógica del proyecto, y debe dar una valoración de la calidad del producto. La aplicación de una lista de comprobación puede durar entre una hora y dos horas. El juicio del líder del equipo, generalmente el Jefe de aseguramiento de calidad, se acepta sin la evidencia de examinar los productos de la práctica o del trabajo; sin embargo, la evidencia, tal como la documentación, que podría ser proporcionada se debe anotar explícitamente en la lista de comprobación. Usando la lista de comprobación en intervalos regulares, el líder de proyecto puede medir el movimiento hacia las prácticas recomendadas. En general, las preguntas de las Listas de Comprobación cubren:

- Compromiso del proyecto con las metodologías o métodos;
- La capacidad del proyecto de satisfacer los estándares;
- Prácticas y actividades relacionadas;
- Seguimiento de los objetivos del proyecto.
- Detección de los defectos cometidos en el ciclo de vida.
- El crecimiento paulatino de la madurez en el proceso
- El aumento paulatino de la cultura en el desarrollo de software de los empleados.

Es por todo esto que las Listas de Comprobación son una herramienta para las Inspecciones de Software, que está al alcance de las organizaciones que inician la transformación hacia un nivel superior de organización y de garantía de calidad.

La rentabilidad más grande de las Listas de Comprobación es que ayudan a identificar defectos importantes en tempranas etapas de desarrollo. Un defecto detectado por las Listas de Comprobación debe ser tratado, por lo general, por la Gestión de Software, realizando

una petición de cambio, que puede ser de modificación, extensión o corrección de errores del software.

En el capitulo siguiente se presenta una clasificación de las Lista de Comprobación, explicando cual es el propósito de cada una de ellas, divididas en tres partes y en el Anexo A, se presenta, con detalle, un conjunto de las listas que ayudaran a la planificación de nuevas inspecciones y a la creación de nuevas listas de comprobación para los proyectos.

Una lista de comprobación esta conformada por varios objetivos o atributos de calidad, cada uno de ellos contiene una serie de preguntas cerradas, ya que una lista de comprobación es estricta en la verificación [32], el cumplimentar parcialmente un requisito no implica su satisfacción. Un SI o un NO son las posibles respuestas por cada una de las preguntas, estableciendo una conformidad al contestar afirmativamente y un defecto al contestar negativamente. Al encontrar un defecto y contestar la pregunta negativamente debe existir una explicación detallada del por qué del defecto, de esta forma se obtiene una lista de los defectos dentro del artefacto.

Dando un peso de importancia a cada una de las preguntas y objetivos dentro de una lista de comprobación, se puede establecer una valoración de calidad. Por cada una de las preguntas y objetivos que se tenga dentro de la lista de comprobación se tiene un peso de importancia entre 0 y 10 de acuerdo a la importancia para el Jefe de Aseguramiento de Calidad o Coordinador. Se explicó que cada una de las preguntas son cerradas, con respuestas SI (conformidad) y NO (no conformidad, existencia de un defecto). Al establecer una respuesta NO se pierde automáticamente el peso de la pregunta que tiene la respuesta negativa.

El cálculo de la valoración de lista es el resultado del cociente, de la sumatoria del producto de los pesos de los objetivo por la sumatoria de los pesos de las preguntas con respuesta afirmativa (SI), entre la sumatoria del producto de los pesos de los objetivos por la sumatoria de todos los pesos de cada pregunta.

$$V_{Total} = \frac{\sum_{i=1}^{n}\left(P_{obj}^{i}\left(\sum_{j=1}^{m}P(si)_{preg}^{i,j}\right)\right)}{\sum_{i=1}^{n}\left(P_{obj}^{i}\left(\sum_{j=1}^{k}P_{preg}^{i,j}\right)\right)} * 100$$

V_{Total} Valoración total de la lista de comprobación

P_{obj} Peso del objetivo

P_{preg} Peso de la pregunta

Esta valorización esta expresada en un por ciento para un mayor entendimiento.

La ayuda de las Listas de Comprobación para la mejora del proceso de Inspección de Software y para todo el proceso de desarrollo, se debe a la posibilidad de realizar un análisis de los resúmenes de defectos encontrados mediante su utilización, generándose de esta forma propuestas de cambios en el proceso para implantarlos en posteriores inspecciones. La valoración de de los resultados de una inspección donde se utiliza listas de comprobación, ayudará a determinar si el proceso en general ha llevado de acuerdo con lo planificado y si los desarrolladores han invertido correctamente el tiempo para el desarrollo.

El formato de una Lista de comprobación, ver el Anexo B, lleva la información siguiente: un Nombre de Proyecto, el Artefacto a inspeccionar, Nombre del Inspector o Inspectores que valoraran la lista y por cada pregunta existente en la lista debe existir cuatro casillas para la valoración del inspector, que puede ser SI (conformidad), NO (no conforme), NP (no procede con el artefacto) y las observaciones, que es una breve explicación de la no conformidad. Existe un dato que no es de interés del inspector, que es el peso de importancia que se le da a cada una de las preguntas y objetivos, explicados anteriormente, este peso no debe ser conocido por el inspector.

1.6 LISTAS DE COMPROBACIÓN

Las Listas de Comprobación son una herramienta de las inspecciones para el control de calidad en muchas situaciones; son un recurso primario en ambientes de la ingeniería, y constituyen un recurso de referencia, preparación y actualización para los inspectores. Una

lista de comprobación se puede mirar como un sistema de reglas fijadas que debe haber sido aplicadas en la creación de algún producto o de un documento que describe el software. Las ediciones se diseñan para ayudar a inspectores a reconocer violaciones evidentes de las reglas específicas.

Antes de dar un concepto más general de las Listas de Comprobación se analiza las características y clasificación de los defectos que se pueden detectar en las inspecciones con la ayuda de las listas de Comprobación.

1.6.1 DEFECTOS [10]

Los errores o defectos son cosas incorrectas que comenten las personas, sin tener en cuenta cuándo y quién los comete, los defectos son elementos defectuosos del software causados por un error. Un error lo comete una persona cuando ejecuta una acción. Así, las personas cometen errores o equivocaciones mientras que los programas tienen defectos. Cuando los ingenieros cometen errores que conducen a defectos, se denomina "introducción de defectos". Esto significa que para reducir el número de defectos que se introduce en los productos, se debe cambiar la forma de hacer las cosas. Para eliminar los defectos en los productos, sencillamente hay que encontrarlos. "La eliminación de los defectos es, por tanto, un proceso más sencillo que la prevención de defectos. La prevención de defectos es un aspecto importante y prioritario que requiere un estudio comprensivo de todo el proceso del software" [26]. Para cometer pocos defectos se debe aprender de los defectos e identificar los errores que los causan.

Un defecto, en su sentido más amplio, es una anomalía en la especificación, diseño o implementación de un producto. Una mejora no es un defecto. Se entiende por mejora a una solicitud una vez obtenido el producto ya que se tuvo en cuenta su funcionalidad en el desarrollo del producto. A continuación se presenta una posible clasificación de los defectos en el desarrollo de software, para cualquier momento del desarrollo de software.

1.6.1.1 **DEFECTOS EN ESPECIFICACIONES / REQUISITOS** [10]

Es un defecto en la definición de las necesidades del usuario para el sistema o componente.

- Requisitos o especificaciones: Descripción mala de los requisitos de los usuarios.
- Funcionalidad: Características del sistema incorrectas o incompatibles.
- Interfaz de Usuario, Software y Hardware: Especificaciones incorrectas sobre como el producto interacciona con el entorno.
- Descripción funcional: Descripciones mal hechas sobre que hace el producto.

1.6.1.2 DEFECTOS DE DISEÑO [10]

Es un error en el diseño del sistema o de un componente. Los errores pueden ocurrir en algoritmos, lógica de control, estructuras de datos, acceso a bases de datos, formularios de entrada y salida, descripción de la interfaz. Estos errores pueden causar un incorrecto código en la construcción.

- Hardware, software e interfaz de usuario: Errores con el producto sobre la interacción del entorno con el usuario.
- Descripción Funcional: El diseño no hace lo que el módulo o producto debiera. Son defectos encontrados durante la inspección del diseño o durante la implementación.
- Comunicaciones entre procesos: No existe comunicación entre interfaz o módulos del sistema en desarrollo y se pierden los datos.
- Definición de datos: Diseño incorrecto de las estructuras de datos que se utilizarán en los módulos del sistema en desarrollo.
- Diseño del módulo: Problemas con el flujo de control y ejecución entre procesos.
- Descripción de la lógica: el diseño es incorrecto en la lógica comparada con el análisis. Es un defecto encontrado durante la inspección o la implementación.
- Chequeo de errores: incorrecta verificación de las condiciones de error dentro del sistema en desarrollo.
- Estándares: el diseño no cumple con los estándares internos.

1.6.1.3 DEFECTO DE CÓDIGO [10]

Son errores causados por una pobre comprensión del diseño o mala elección de las estructuras de datos y algoritmos, o errores de lógica o sintaxis. Algunos errores no pueden

ser detectados a menos que se utilicen datos de pruebas adecuados o algún método de inspección de código.

- Errores o equivocaciones en la implementación de un programa: pueden estar en el producto, en el código de prueba, ficheros, etc.
- Lógica: pasos olvidados, lógica duplicada, funciones innecesarias, etc.
- Problemas de computación: pérdida de precisión, ecuaciones incorrectas, etc.
- Problemas de manipulación de datos: inicialización incorrecta, acceso o almacenamiento de datos incorrecto, unidades o escalas incorrectas, incorrecta dimensión de lo datos, etc.
- Implementación / interfaz del módulo: problema relacionado con llamadas, paso de parámetros, terminación de subprocesos, etc.
- Estándares: el código no cumple con los estándares internos.

1.6.1.4 DEFECTOS DE DOCUMENTACIÓN [10]

Errores en manuales, instrucciones de instalación, demostraciones, todos ello centrado al cliente.

1.6.1.5 DEFECTOS DEL ENTORNO DE APOYO [10]

Son los que resultan del entorno de desarrollo y pruebas, como errores de configuración, de integración de herramientas, etc.

- Software de pruebas: problemas de software utilizado para probar las capacidades del producto.
- Hardware de pruebas: problemas del hardware de pruebas, no del hardware en el que corre el producto.
- Herramienta de desarrollo: si no se comportan adecuadamente
- Software de integración: problemas de interacción software/herramientas.

1.6.1.6 MODOS [10]

Defectos que se encuentran cuando existen varias maneras de realizar el proceso o artefacto y no se tomo la más adecuada.

- Olvidado: la información no aparece en el producto intermedio.
- Confuso: la información es engañosa, ambigua o difícil de entender.
- Incorrecto: la información es claramente incorrecta.
- Cambio: cambios en el producto intermedio provocaron cambios en otro producto intermedios.
- Mejor manera: existía otra mejor manera de realizar ese producto intermedio, y por motivos de eficiencia, rendimiento, legibilidad, facilidad de mantenimiento u otros no se realizó.

1.7 CONJUNTO DE REGLAS PARA LAS LISTAS DE COMPROBACIÓN

Este conjunto de reglas deben emplearse para la planificación de una nueva inspección, tratando de no violar ninguna de ellas. Pero, al aplicar a una empresa nueva todo el proceso de inspección de software o que la empresa solo tenga el objetivo de mejorar el proceso de desarrollo de software, las listas de comprobación puede violar este conjunto de reglas, pero no por mucho tiempo, ya que, a medida que el proceso se retroalimente por la detección de los defectos, las reglas serán cumplidas satisfactoriamente [51].

- Una lista de comprobación debe derivarse del sistema de reglas generales del producto en desarrollo.
- Cada pregunta de la lista de comprobación debe incluir una referencia a la etiqueta de la regla que interpreta dentro de un estándar o reglamentos internos de la empresa desarrolladora.
- Las listas de comprobación deben ser actualizadas cuando sea necesario para reflejar el descubrimiento de los defectos importantes no incluidos hasta este momento.
- Las preguntas y objetivos para una lista de comprobación no debe exceder de una sola página física.
- Las descripciones de las preguntas y objetivos de la lista de comprobación pueden incluir las sugerencias para la severidad probable del defecto.

- Esta generalmente redactada cada pregunta, pero no necesariamente, para que una respuesta negativa identifique un defecto.
- Las preguntas de la lista de comprobación deben concentrarse en divulgar defectos importantes.
- Los resultados de la lista de comprobación no debe emplearse de mala forma para definir una nueva regla.

1.8 SOFTWARE EXISTENTES. [45]

En la actualidad no hay en Cuba la existencia de un software que ayude al proceso de las Inspecciones de Software. Por este motivo se realiza una búsqueda exhaustiva en Internet para ver si existe un software que nos ayude a construir el sistema propuesto de este trabajo. Se encontraron distintos tipos, con características diferentes, se especifican algunos de ellos, los cuales han servido para comparar las distintas funcionalidades del nuevo software para el Proceso de Inspección.

1.8.1 INSPECCIÓN INTELIGENTE DE CÓDIGO EN AMBIENTE DE LENGUAJE C (ICICLE, INTELLIGENT CODE INSPECTION IN A C LANGUAGE ENVIRONMENT)

ICICLE, el nombre sugiere que es un ayudante inteligente automatizado para la inspección del código de C y C++. Esta herramienta de inspección es única y hace uso de conocimiento para asistir a encontrar defectos comunes. Ya que el conocimiento está en una clase muy específica, que controla la base de lenguaje C, ICICLE no es conveniente para apoyar a una inspección general. Puede, sin embargo, ser utilizado para examinar archivos llanos de texto, por ejemplo documentos de Microsoft Word.
La herramienta esta diseñada para apoyar dos fases de la inspección: la preparación (individual) y la reunión de la inspección. Durante la reunión de la inspección, la herramienta proporciona la funcionalidad de la comprobación individual.

Características.

- El código fuente se exhibe en una ventana grande con cada línea numerada.
- Prepara automáticamente comentarios sobre código fuente usando sus herramientas de análisis.
- Se puede utilizar para señalar defectos en una línea específica del código fuente.
- Capacidad de modificar e incluir procedimientos para requisitos particulares del análisis.
- Proporciona la información de los objetos tales como variables y funciones en C.
- Genera una lista de todos los defectos encontrados.
- Genera un resumen de los defectos por tipo, clase y severidad.
- Prepara un informe que detalla el tiempo total pasado en la preparación y en la reunión.

1.8.2 INSPECCIÓN DE SOFTWARE COLABORATIVA (CSI, COLLABORATIVE SOFTWARE INSPECTION)

Esta diseñado para apoyar la inspección de todos los productos del desarrollo del software. La herramienta se describe en relación al modelo de inspección de Humphrey. En esta variación, cada inspector crea una lista de los defectos durante la inspección individual, que se da al autor del documento antes de la reunión de la inspección. Es la tarea del autor correlacionar estas listas individuales y después tratar cada defecto en la reunión de la inspección.

Características.

- CSI proporciona un navegador para ver el documento bajo inspección, que numera automáticamente cada línea.
- Puede anotarse que es lo que falta en el artefacto.
- El inspector puede categorizar y clasificar los defectos.
- Presenta un resumen al autor clasificando cada defecto y puede aceptar o rechazarlo.

- Se puede tener un acceso a todas las anotaciones asociadas al documento y correlacionarlas en una sola lista de defectos.
- Permite al autor clasificar la lista de defectos en opciones múltiples, incluyendo severidad, la época de la creación y el estado actual.
- Prevé la inspección distribuida, permitiendo una inspección con los miembros del equipo en una variedad de lugares.
- Proporciona un registro de la historia de la inspección.
- Recoge varias métricas del proceso, tal como el tiempo que pasa en la reunión y el tiempo que tarda en encontrar un defecto, así como el número y severidad de los defectos encontrados.

1.8.3 EXAMEN CUIDADOSO (Scrutiny)

Con la herramienta Scrutiny se puede realizar una inspección en general de los productos en el ciclo de vida del software. Permite la inspección distribuida, puede ser integrado con las herramientas existentes y puede ser adaptado para apoyar diversos procesos del desarrollo. Se basa en cuatro etapas. La primera etapa se llama iniciación y es comparable a la del modelo de Fagan. La segunda etapa es la preparación, como en el modelo de Fagan. La reunión, a sí mismo como en Fagan, está en la etapa final, que abarca la reanudación y un resumen. Los roles por cada participante son también similares, no obstante el Scrutiny también pone algunos cambios en la ejecución. Primero, el rol del modelador se puede cambiar o incluir los deberes del lector. Además, el rol del registrador se puede tomar por más de una persona. El examen pone, explícitamente, el rol del productor, que puede contestar a preguntas con respecto al documento. Finalmente, hay otro rol, verificador, asegura que los defectos encontrados han sido tratados por el autor y por el equipo de inspección. Este papel se puede asignar a cualquier participante. Cada etapa del proceso, se modela en Scrutiny.

Características.

- Existe una ventana donde se puede visualizar el documento que está en inspección.

- Apoya, solamente, a documentos de texto.
- No utiliza la Lista de Comprobación, ni otra documentación de soporte.
- Todos los defectos se encuentran manualmente.
- Cada inspector tiene la oportunidad de agregar una hora preferida para cualquier preparación.
- Se ha probado con inspecciones locales y distribuidas.
- Se ha utilizado instalaciones de audio y de tele conferencia.
- Un inspector puede enviar puntos de discusión al resto de los participantes.
- Proporciona medios de enviar mensajes simples a los participantes de la reunión.
- Genera automáticamente un informe de la inspección que contiene toda la información relevante sobre la inspección y sus participantes.
- Obtiene detalles del tiempo pasado por cada participante en la inspección y el cubrimiento del documento que alcanzaron en el transcurso del tiempo empleado para la inspección.
- Obtiene Una lista completa de defectos con la información necesaria.

1.8.4 INSPECCIÓN DE SOFTWARE EN FASES PARA ASEGURAR LA CALIDAD (InspeQ, INSPECTING SOFTWARE IN PHASES TO ENSURE QUALITY)

InspeQ, es un conjunto de herramientas desarrollado por Knight y Myers, para apoyar su técnica propuesta para la inspección. La técnica fue desarrollada por Knight y Myers con la meta de permitir que el proceso de la inspección sea riguroso, justo, eficiente en su uso de recursos, y apoyada por la computadora.

A pesar de esto, entre los sistemas existentes en la actualidad, el InspeQ esta considerado como uno de los que menos soporte brinda a la gestión de documentos.
Si posee un desempeño favorable en la etapa de la preparación individual asistiendo al trabajo del inspector mediante el uso de listas de comprobación y asegurándose que cada punto de la lista es vista por el inspector. Permite además el uso de documentos que representen estándares, lo cual también viene a ayudar en la etapa de preparación individual.

Características

- Existen dos tipos de Fases: solo un inspector y múltiples inspectores.
- Se utiliza una lista de comprobación rigurosa.
- Las inspecciones en fase se diseñan para permitir que los expertos se concentren en encontrar defectos.
- El inspector puede examinar simultáneamente partes separadas del mismo documento.
- InspeQ realiza el formato de estos comentarios antes de que se pasen al autor.
- Asegura de que todos los artículos de la lista de comprobación son tratados por el inspector antes de las salidas del producto y la fase.
- Apoya la supervisión de personal.
- Genera una lista de comentario para cada inspector.

1.8.5 SISTEMA DE COLABORACIÓN A LA REVISIÓN DE SOFTWARE (CSRS, COLLABORATIVE SOFTWARE REVIEW SYSTEM)

El sistema de colaboración a la revisión de software (CSRS) es un ambiente para apoyar el uso de FTArm (método asincrónico técnico formal) de la revisión, un desarrollo obtenido a partir del método de inspección de Fagan.

Características.

- es un método general para examinar cualquier tipo de documento.
- El informe de la inspección final es producido por el asesor.
- Un documento se almacena en una base de datos como una serie de nodos interrelacionados. Para el código de fuente, estos nodos consistirían en funciones y otras construcciones del programa.
- Las anotaciones también se almacenan como nodos de tres tipos: nodo del comentario, nodo de la edición, nodo de la acción.
- Proporciona una lista de comprobación en línea para asistir al inspector.

- Proporciona la colección automática de los datos tales como número y la severidad de los defectos.
- También tiene la capacidad de guardar un registro de los acontecimientos.

1.9 RESUMEN DE LAS CARACTERÍSTICAS DE SOFTWARE EXISTENTE

Se puede ver las siguientes características en el software existente:

- Todas las herramientas describen los documentos de texto adecuadamente.
- ICICLE, el Scrutiny y CSI utilizan la misma técnica de exhibir el documento con cada línea numerada.
- CSRS divide el documento en pedazos más pequeños llamados nodos.
- Inspe Q tiene los comentarios fuera del documento fuente.
- ICICLE, CSI, el Escrutinio y CSRS todos permiten la clasificación de los defectos y sus anotaciones, mientras que InspeQ permite solamente su creación o cancelación.
- Las listas de comprobación son apoyadas solamente por InspeQ.
- CSI tiene el concepto de una lista de los criterios que ayude a los inspectores a encontrar y a categorizar los defectos.
- InspeQ exhibe estándares, mientras que ICICLE puede proporcionar una facilidad de hojear las páginas de los manuales dentro del software.
- ICICLE es la única herramienta para proporcionar cualquier detección automática del defecto para código en C o C++.
- CSRS puede proporcionar los detalles de la cantidad de tiempo que pasa un inspector al momento de hacer la inspección.
- El Scrutiny almacena el porcentaje del documento cubierto por cada inspector, también el tiempo pasado por cada inspector en la preparación y la reunión.
- CSI utiliza Tele conferencia, que proporciona un canal audio para la discusión entre los inspectores.
- El Escrutinio también apoya el uso de un canal audio para la discusión entre los inspectores, además de sus instalaciones para la discusión y la mensajería.

- ICICLE recolecta automáticamente métricas en el número y tipo de comentarios hechos, así como su severidad, según lo observado por el anotador durante la reunión de la inspección.
- CSI utiliza un registro para registrar la métrica del defecto incluyendo la severidad, tiempo tomado para encontrar el defecto e intervalo de tiempo total pasado en la reunión.

1.10 CONCLUSIONES

Durante el capitulo se presento una vista general de las Inspecciones de Software y se puede verificar que estas ayudan al aseguramiento de calidad.

La calidad total de un producto es muy difícil obtenerla y más aun para el software, solo se la puede maximizar. La calidad total esta fundamentada en diversos puntos de la Ingeniería de Software como se ha visto. Uno de los más importantes y que no debe faltar, es una herramienta para detectar los defectos en cualquier momento del desarrollo y cuantificar con la ayuda de las métricas a dar un significado al proceso que se sigue, además, realizar un seguimiento de los distintos atributos de calidad que fueron definidos para el proyecto de software en los requisitos no funcionales. Esto permite caracterizar al proceso de desarrollo, permite tener bases objetivas para mejorar el proceso de software.

Se ve que los atributos de calidad, los métodos de inspección, las listas de comprobación y las métricas definidas para las inspecciones, son un conjunto de herramientas que permiten la mejora incremental del proceso de desarrollo en el transcurso del tiempo y así lograr cumplir con las recomendaciones de los Modelos de Madurez de Capacidad CMMI y CMM.

Se logro conocer las capacidades de los software existentes, dando un punto de partida para la creación del nuevo Soporte al Proceso de Inspecciones de Software.

CAPITULO 2 MODELO DE REFERENCIA PARA LA INSPECCIÓN

2.1 INTRODUCCIÓN

El presente capítulo explica el modelo que el autor considera para la puesta en práctica de las inspecciones de software en el entorno de producción industrial de software, se describe las etapas del modelo y la información que se captura y se genera a través del modelo.

También, se muestra la clasificación, que considera el autor, de las listas de comprobación que se agrupan en tres: de Proceso de Software, Ciclo de Vida y Atributo de Calidad. Entendiéndose como, Proceso de Software a las etapas de planificación y control del proceso de desarrollo (por ejemplo, Plan de Aseguramiento de Calidad, Plan de Gestión de Configuración de Software, Pre-Aceptación del Cliente). Se entiende por Ciclo de Vida como las etapas centrales de desarrollo de software (por ejemplo, Análisis, Diseño, Implementación, Prueba). Se entiende por Atributos de Calidad como las facilidades y propiedades que requiere el software para cumplir con los requisitos propuestos antes de su desarrollo. Además, se da una explicación de la utilización de cada una de las listas de comprobación que parecen en la clasificación.

2.2 MODELO PARA LA REALIZACIÓN DE LAS INSPECCIONES

Como consecuencia del estudio y análisis de los distintos métodos y analizando las dificultades en la práctica de las inspecciones de software, el autor ha llegado al siguiente modelo de inspección de software el cual ha servido de base para la automatización y está basado en el modelo de Fagan y Tom Gilb.

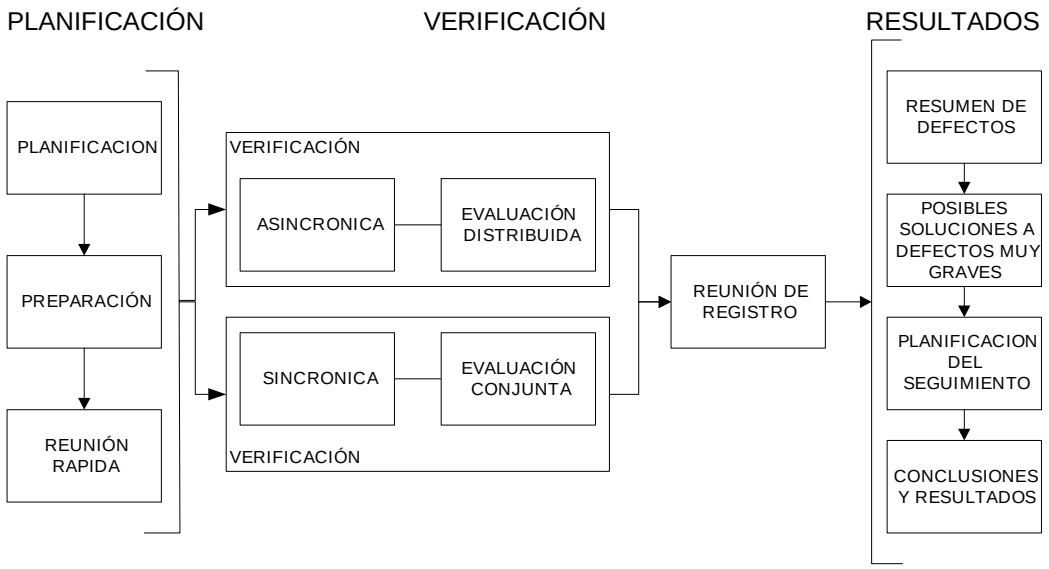


Figura 2.1. Modelo de Referencia para la Inspección

El Proceso de Inspección se inicia con la Planificación, luego de una reunión del Jefe de Aseguramiento de Calidad o Coordinador y el Desarrollador, la cual es la etapa más importante donde se planifica toda la información que guiará a toda la inspección, para luego pasar a la Preparación de los diferentes inspectores. Los Inspectores luego de ser escogidos en la etapa anterior, deben prepararse de acuerdo con los documentos establecidos en la planificación, verificar los estándares y reglas empleadas en el desarrollo. Estos documentos de preparación juntamente con los objetivos y propósitos de la inspección son revisados en la Reunión Rápida, donde por primera vez se encuentran los integrantes del equipo de inspección, la reunión no debe ser muy larga y es dirigida por el Jefe de Aseguramiento de Calidad o el Coordinador. Cuando termina esta última etapa, Reunión Rápida, los inspectores están listos para realizar la verificación, ya sea asincrónica o sincrónica, dependiendo de las características y condiciones del software, en esta etapa se aplican las Listas de Comprobación y se realiza la captura de los defectos, los cuales se resumirán en una Reunión de registro, donde algunas veces participará el Desarrollador para explicar y entender mejor los defectos. De esta Reunión de Registro sale un Resumen de defectos que son clasificados por su gravedad e importancia, esta clasificación es responsabilidad del Jefe de Aseguramiento de Calidad. En algunos casos, para los defectos muy graves se realiza una reunión para encontrar soluciones, donde participan expertos en un determinado campo. La penúltima etapa es la Planificación del Seguimiento de los

defectos, donde el Jefe de Aseguramiento de Calidad será el responsable de ir eliminando los defectos ya editados o realizará una petición de cambio, para finalizar el proceso de inspección. También, se elabora a través de las métricas un resumen de resultados, que ayudará a mejorar el proceso de inspección.

En todo el proceso participan distintas personas (Actores) como: Jefe de Aseguramiento de Calidad o Coordinador, Asesor, Jefe de Inspección, Lector, Anotador, Desarrollador y los Inspectores.

2.3 ROLES EN LA INSPECCIÓN

Como se señala en el anterior epígrafe, existen distintas personas que participan en el proceso de inspección de software.

En una empresa pequeña, es recomendable que exista el Jefe de Aseguramiento de Calidad, que puede formar parte de los inspectores, así como el Asesor y un Inspector, que pueden ser otras personas que conforman la empresa. Con estos tres roles puede llevarse a cabo la inspección, repartiendo entre ellos los demás roles.

Para mayor conocimiento de los roles en el proceso de inspección de software, se detallan las actividades que deben cumplir cada uno de estos:

Jefe de Aseguramiento de Calidad o Coordinador. Debe cumplir las siguientes tareas en el proceso de inspección:

- Verificar que el producto cumple los criterios de entrada. Si no los cumple, tendrá que indicarle al autor que es lo que le falta al producto para poder ser inspeccionado. Esta verificación previa sirve para asegurar que el producto esté preparado para su revisión.

- Determinar la necesidad de una sesión de adiestramiento. Es necesaria si los inspectores tienen un conocimiento insuficiente del producto o de los estándares empleados en la empresa.
- Seleccionar a los demás inspectores y al asesor, negociando los roles de cada uno de estos, de forma que queden equilibrados los puntos de vista técnicos citados anteriormente sobre los cuales debería revisarse el producto.
- Programar la fecha, hora y lugar de las reuniones.
- Preparar y distribuir la notificación de la inspección a todo el equipo. En esta notificación se indica el tipo de inspección (o presentación), la fecha, hora, lugar, duración estimada de la reunión, y el tiempo de preparación estimado para los participantes, estándares a utilizar y reglas.
- Organizar, anunciar y conducir la reunión de presentación.

Asesor. Debe cumplir las siguientes tareas en el proceso de inspección:

- Ayudar a la preparación de la inspección. El Jefe de Aseguramiento de calidad debe apoyarse en los conocimientos del asesor para elaborar los documentos y listas de comprobación necesarias para la inspección.
- Debe apoyar al grupo de inspectores en el momento de la inspección.
- Debe colaborar con la elaboración del resumen de defectos en la reunión de registro.

Anotador. Debe cumplir las siguientes tareas en el proceso de inspección:

- Es parte de los inspectores.
- Debe llevar un resumen detallado de los posibles defectos encontrados en la etapa de verificación para luego discutirlos en la reunión de registro.
- En la reunión de registro debe elaborar el resumen de defectos que será entregado al desarrollador y al Jefe de Aseguramiento de Calidad.

Lector. Debe cumplir las siguientes tareas en el proceso de inspección:

- Es parte de los inspectores.
- Leer las listas de comprobación en la etapa de verificación.
- Ayuda al Jefe de inspección a velar los estándares y reglas durante la verificación.

Desarrollador. Debe cumplir las siguientes tareas en el proceso de inspección:

- Recopilar todos los documentos necesarios para la inspección del artefacto junto con el moderador para verificar que cumplan los criterios de entrada, y dar una copia al moderador del artefacto a inspeccionar.
- Facilitar y distribuir la documentación al resto de los participantes.
- Recomendar o no la realización de una sesión de presentación y explicación del producto.
- Discusión de los defectos encontrados en la reunión de registro.

Jefe de inspección. Debe cumplir las siguientes tareas en el proceso de inspección:

- Debe llevar a cabo la reunión de registro viendo la participación del desarrollador.
- Apoyar a los inspectores en la detección de los defectos.
- Verificar que se cumplen los estándares y reglas establecidas para la inspección.
- Elaborar juntamente con el anotador el resumen de defectos.
- Verificar que se cumpla la agenda planificada (horas y fechas).

Inspector. Debe cumplir las siguientes tareas en el proceso de inspección:

- Estudiar el material y documentación de apoyo entregado, durante un tiempo estimado indicado en la notificación de la inspección.
- Utilizar listas de comprobación para detectar defectos.
- Si tiene el papel de lector, preparar cómo va a realizar la lectura del producto.

- Anotar el tiempo empleado de preparación, para comunicarlo al moderador al inicio de la reunión. Este dato se utilizará para evaluar la efectividad del proceso de inspección.

Cada una de estas personas juega un papel muy importante dentro de las Inspecciones de Software. En la tabla 2.1 se observa la participación de los actores en las distintas fases de la Inspección de Software.

Etapa	Jefe SQA	Asesor	Anotador	Lector	Desarro-llador	Jefe de Inspección	Inspector
Planificación	X	X			X		
Preparación	X	X				X	X
Reunión Rápida	X		X	X		X	X
Verificación		X	X	X		X	X
Reunión de Registro		X	X			X	
Resumen de Defectos		X			X	X	
Posibles Soluciones		X			X	X	
Planificación del Seguimiento	X						
Conclusiones y Resultados	X						

Tabla 2.1 Participación de los Actores en la Inspección de Software. “X” es necesaria la inclusión del participante en la etapa del modelo.

2.4 DESCRIPCIÓN DEL MODELO

Para comprender el modelo propuesto para las inspecciones de Software, se describe a continuación cada etapa. Señalamos que se divide en tres fases, la primera fase es de

Planificación, que está conformada por Planificación, Preparación y la Reunión Rápida, es la etapa donde se define las tareas, documentos y las personas que participarán, para luego pasar a la parte de la Verificación, que esta conformada por la Verificación y la Reunión de Registro, donde se aplica las listas de comprobación para obtener una valoración del artefacto, y la última etapa que son los Resultados y Conclusiones, que está compuesta por Resumen de Defectos, Posibles Soluciones, Planificación del Seguimiento y las Conclusiones y Recomendaciones.

A continuación, se desarrolla cada fase del modelo propuesto, explicando la participación de los actores, la información necesaria y generada en cada una de ellas, y los pasos que se deben seguir para continuar con la siguiente fase.

2.4.1 PLANIFICACIÓN

El autor considera que la etapa de planificación es la parte más importante en las inspecciones de software, es donde se define la información requerida para llevar acabo todo el proceso. En los tres modelos estudiados en el anterior capitulo, se ve que cada uno de estos modelos lleva una etapa de planificación [20][21][22], lo cual muestra la importancia dentro del proceso de inspección de software. El autor considera que la etapa de Planificación debe existir en el nuevo modelo.

Antes de llevar acabo la etapa de planificación, el Jefe de aseguramiento de Calidad o Coordinador conjuntamente con el Desarrollador verifica si el artefacto esta correctamente terminado y con la información necesaria para realizar la inspección. El autor considera necesaria la siguiente información:

- Se selecciona a los inspectores, con su respectivo rol (cargo que desempeñará) y perfil (línea de inspección) respecto a la especialidad y participación que ha tenido en anteriores inspecciones. La primera persona que se seleccionará será el asesor, con él se definirá a los demás inspectores que participarán en la inspección, designando un Jefe de Inspección. Los inspectores no deben sobrepasar el número de cinco incluyendo al asesor y al Jefe de Inspección y no deben ser menos de tres.

- Se debe describir el Proyecto el cual utilizara las inspecciones para verificar el artefacto desarrollado, teniendo en cuenta la siguiente información: El Nombre del proyecto y una breve explicación de los objetivos, metas que quiere alcanzar al culminar el desarrollo y el tipo de software.
- Debe describirse la inspección tomando en cuenta la siguiente información: el nombre del artefacto, la etapa que se encuentra el proyecto, estándares a utilizar, si se llevará acabo sincrónicamente o asincrónicamente, la fecha de inicialización y finalización, y el propósito de la inspección. Lo último son reglas de partida, las cuales servirán para la preparación de los inspectores.
- Debe definirse los Documentos de Apoyo que servirán para la preparación de los inspectores, dando a conocer los puntos que deben tomar en cuanta cada uno de los inspectores. Los documentos de apoyo, por ejemplo, serán los estándares internacionales como normas ISO o estándares específicos de la empresa desarrolladora o documentos que describen una estructura estándar de desarrollo, que debe ser conocida, del artefacto que quiere ser inspeccionado.
- Debe ser definida cada una de las etapas que conforma el modelo, para su ejecución, con la siguiente información: Hora de ejecución y fechas límite para los encuentros y reuniones y además el lugar donde se llevara a cabo.
- Debe seleccionarse las listas de comprobación que acompañara a la inspección de software, de acuerdo al propósito, el tipo de software, la etapa de desarrollo y el tipo artefacto. La experiencia del Jefe de Aseguramiento de Calidad y el asesor será algo importante para la determinación de las listas de comprobación.
- Debe definiese los pesos correspondientes a cada pregunta y objetivo respecto a los objetivos de la inspección. El Jefe de Aseguramiento de Calidad debe definir la importancia, a su parecer y experiencia, el peso y objetivo asociado a cada pregunta que conforma la Lista de Comprobación.

En la planificación participa Jefe de Asuramiento de Calidad y un Asesor, que luego se convierte en un inspector participando en la verificación.

2.4.2 PREPARACIÓN

Una vez planificada la inspección y haber definido a las inspecciones con sus roles a seguir. Estos necesitan un tiempo para la lectura de los documentos de apoyo de la inspección, teniendo como principal objetivo el conocimiento total de lo que se va a inspeccionar. De esta forma estarán preparados e informados sobre el artefacto que se va a inspeccionar. Como señala el modelo de Fagan en la etapa de Preparación [21].

Esta etapa de Preparación, no es muy importante, pero si se recomienda que el inspector consuma tiempo en la lectura de los documentos para conocer los estándares y consideraciones que se aplicaran a la inspección. De este modo se garantizará, en gran medida, la detección de los defectos en el artefacto a inspeccionar y el inspector no improvisará al momento de de la verificación.
Dentro de la Preparación para la inspección participan el Inspector, el Jefe de Inspección, el asesor, lector, anotador y el Jefe de Aseguramiento de Calidad. Estos roles generan la siguiente información que el autor considera necesarios.

- Se debe capturar el Tiempo de Preparación por cada inspector.
- Deben estar de acuerdo, los inspectores, sobre la planificación de la agenda. Los participantes en la inspección tienen la opción de realizar observaciones a la agenda planificada.
- Deben, los inspectores, leer las listas de comprobación y estar de acuerdo con estas, si no pueden realizar observaciones sobre cada una.

2.4.3 REUNIÓN RÁPIDA

El primer encuentro del equipo de inspección es una Reunión Rápida, que debe estar planificada. Se aclaran puntos diversos que no han sido entendidos en la documentación de apoyo y puntos olvidados que no se han tomado en cuenta en la planificación.

Esta reunión, debe durar alrededor de los 15 minutos, que es suficiente para la discusión de los objetivos y propósitos de la inspección.

Como señala el modelo de Tom Gilb [21], “el propósito fundamental de la reunión es conocer los objetivos de la inspección y como se la va a realizar”. El autor considera que esta etapa no es muy importante, sin embargo es recomendable solo por el hecho de aclarar diversos puntos junto con los inspectores y el Jefe de Aseguramiento de Calidad.

En esta etapa participan el Jefe de Aseguramiento de Calidad, Lector, Jefe de Inspección y los Inspectores.

La información que se genera será exclusivamente de la inspección, que se lleva acabo en ese momento y es la siguiente:

- Se debe modificar la agenda, si es necesario, en aspectos tales como la corrección del tiempo en el que se hará la verificación, en el caso de que por algún motivo que este fuera del alcance de los inspectores (por ejemplo, mala preparación de algunos inspectores o enfermedad).
- Se puede realizar, en esta etapa del modelo, la eliminación o adición de documentos de apoyo. Como se dijo anteriormente, serán utilizados para la preparación de los inspectores o para la consulta en el momento que se realiza la verificación. El inspector consultará los documentos para asegurar que el artefacto cumpla con las normas planificadas, estos estándares pueden ser internos o internacionales respecto al tipo de software o la etapa de desarrollo.
- Se puede adicionar o modificar las Listas de Comprobación a ser utilizadas. La corrección de una Lista de Comprobación será hecha por el Jefe de Aseguramiento de Calidad con fundamentos claros realizados por los inspectores.

2.4.4 VERIFICACIÓN ASINCRÓNICA Y EVALUACIÓN DISTRIBUIDA

Una de las variantes de las inspecciones es la asincrónica [45], la cual, el autor considera que tiene mucha utilidad cuando el artefacto es muy extenso o cuando el personal de la empresa que desarrolla software es muy limitado. La característica es que el inspector es el que escoge el tiempo de inicialización para la verificación del artefacto, esto quiere decir, que puede iniciar la verificación en cualquier momento o lugar sin tomar en cuenta a los

demás inspectores y es distribuida porque, cada inspector tiene la opción de ver los defectos encontrados por otro inspector en el momento que realiza la verificación. Pero, el inspector, debe iniciar y terminar la verificación en un tiempo establecido, el Jefe de Aseguramiento de Calidad debe hacer cumplir que se lleve de acuerdo con la agenda planificada, para no incurrir en retrasos, ni tampoco en retrasar la inspección.

En este momento es cuando se emplean las listas de comprobación planificadas en la etapa de Planificación, sirviendo al inspector como una guía y recurso para los detalles del artefacto en inspección. El inspector, al efectuar la lectura a cada una de las preguntas de las listas de comprobación y verificando la conformidad de cada una de ellas, efectúa una valoración de acuerdo a su preparación, experiencia y visión, para luego realizar un resumen de los defectos que a su parecer se encuentran en el artefacto. De esta forma, al culminar con la verificación se tiene los posibles defectos que servirán como partida para la Reunión de Registro. También se obtiene una valoración total de la lista por cada inspector, para luego comparar con otros resultados.

La información que se genera será por cada inspector que participe en la verificación, es la siguiente:

- Se obtiene listas de comprobación correctamente verificadas y con los posibles defectos en el artefacto.
- Se obtiene la hora y día de inicio de la verificación por cada inspector.
- Se obtiene el número de defectos y observaciones por cada inspector. Al Finalizar la verificación, se realiza un conteo de los defectos encontrados y por la gravedad, para luego utilizar juntamente con las métricas de las inspecciones de software.
- Se registra el tiempo empleado para la evaluación por cada inspector.
- Se obtiene la valoración de la lista de comprobación por cada inspector. A través de los pesos que el Jefe de aseguramiento de Calidad definió en la etapa de planificación, para cada una de los objetivos y preguntas, se evalúa el artefacto.

2.4.5 VERIFICACIÓN SINCRÓNICA Y EVALUACIÓN CONJUNTA

La Inspección Sincrónica y Evaluación Conjunta, es también una variante de las inspecciones. Todos los inspectores participan en la inspección del artefacto en un mismo tiempo y lugar, encontrando los defectos y comentando cada uno de ellos, es por este motivo que debe asignarse a un Anotador para que vaya tomando nota de todas las observaciones y defectos encontrados a través de las Listas de Comprobación, utilizadas para esta situación. La inspección es dirigida por el Jefe de Inspección.

La información generada es por el grupo de inspección, es la siguiente:

- Se obtiene listas de comprobación correctamente verificadas y con una lista de defectos encontrados en el artefacto.
- Se registra hora y día de inicio de la verificación.
- Se obtiene el número de defectos y observaciones. Al Finalizar la verificación, se realiza un conteo de los defectos encontrados.
- Se registra el tiempo empleado en la verificación.
- Se obtiene la evaluación de cada una de las listas de comprobación. A través de los pesos que el Jefe de aseguramiento de Calidad definió en la etapa de planificación se evalúa cada lista de comprobación.

2.4.6 REUNIÓN DE REGISTRO

Cuando la inspección fue realizada asincrónicamente el Jefe de Inspección, el Asesor y el Anotador realizan un resumen de los defectos encontrados por los demás inspectores, para luego juntamente con el coordinador discutir los defectos y clasificarlos por gravedad. El autor considera que la participación del Desarrollador en esta etapa es muy importante, ya que él va entendiendo la realidad de los defectos encontrados para luego corregir cada uno de ellos. No es obligatoria la participación del desarrollador en esta etapa.

En la inspección sincrónica, ya el Jefe de Inspección, el Asesor y el Anotador tienen el resumen de los defectos encontrados, lo único que falta es la discusión de cada uno de ellos juntamente con el desarrollador y la clasificación según la gravedad del defecto encontrado.

Como menciona Tom Gilb [21], será aceptados o rechazados los defectos y clasificados por su gravedad.

La información que se genera es la siguiente:

- Se obtiene un resumen parcial de defectos.
- Se obtiene una clasificación de los defectos encontrados por la gravedad que considera que tienen.

2.4.7 RESUMEN DE DEFECTOS

Pasada la Reunión de Registro, se obtiene un Resumen de defectos que es considerado por el Jefe de Inspección, el cual juntamente con el Asesor discuten si es necesaria la participación de personas con experiencias o formadas profesionalmente para resolver los defectos muy graves y ayudar al desarrollador a corregirlos.

La información que se genera es:

- Se obtiene un resumen de defectos ordenados por la gravedad para el artefacto inspeccionado.
- Se obtiene una lista de personal adicional para la reconstrucción del artefacto.

2.4.8 POSIBLES SOLUCIONES A DEFECTOS

Una vez que se obtiene la clasificación de los defectos muy graves. Se realiza una reunión con personas con experiencia, dirigida por Jefe de Inspección juntamente con el Desarrollador y el Asesor realizando una Tormenta de Ideas, tratando de ayudar al desarrollador para la corrección de cada uno de los defectos graves.

Como señala Tom Gilb [21], la tormenta de ideas será una gran ayuda y una guía para el desarrollador para la corrección de los defectos.

La información que se genera es la siguiente:

- Se obtiene observaciones de solución por cada defecto grave.
- Se obtiene un informe general de la Inspección.

2.4.9 PLANIFICACIÓN DEL SEGUIMIENTO

De acuerdo con las observaciones por cada defecto. Se planifica un tiempo de reconstrucción y la asignación de las personas que participaran en la reconstrucción. La información necesaria que considera el autor para definir el seguimiento de los defectos es la siguiente:

- Se debe analizar el número de personas a realizar la reconstrucción.
- Se debe especificar el tiempo de reconstrucción.
- Se debe capturar el tiempo real de la reconstrucción por defecto.

Puede existir una petición de cambio, la cual debe ser controlada por la Gestión de Configuración de Software.

2.4.10 CONCLUSIONES Y RESULTADOS

Esta etapa forma parte del modelo de inspección propuesto, es la etapa final.

La Inspección de un artefacto finaliza cuando los defectos encontrados fueron corregidos por el desarrollador. El Jefe de Aseguramiento de Calidad tiene la labor de analizar las métricas para establecer mejoras en el proceso de inspección.

2.5 CLASIFICACIÓN DE LAS LISTAS DE COMPROBACIÓN

La colección, que se presenta, de listas de comprobación, fue obtenida a través de una recopilación de la información obtenida en sitios en Internet relacionados con el desarrollo de software aplicando al proceso de Inspecciones de Software.

Existen muchos sitios Web que se dedican exclusivamente al Aseguramiento de Calidad y dentro de ellos a las inspecciones con las Listas de Comprobación. Como se describe anteriormente puede utilizarse desde el inicio hasta el final de la vida de un software o cuando se definen características o atributos que debe tener el software. Además, al examinar las distintas Áreas de Proceso, dentro de CMM o CMMI, las Listas de Comprobación apoyan a la mayoría de estas. La clasificación que se presenta da una idea de la utilidad y de la aplicabilidad en las Inspecciones. El autor las clasifica en tres grupos: uno, que es sobre el Proceso de Software, otro sobre el Ciclo de Vida y por último, por los Atributos de Calidad según McCall [13]. En el Anexo A se presentan todas las Listas de Comprobación completas para su uso en el desarrollo de Software, ahora se presenta la descripción de cada una de ellas.

2.5.1 PARA EL PROCESO DE SOFTWARE

Pre-Aceptación. [27]

La preparación de la aceptación del sistema de software es para asegurar que todo está en el lugar para comenzar las actividades del proceso de la aceptación. La lista de comprobación de Pre-aceptación ayuda a asegurar que las actividades necesarias han terminado y que los documentos de funcionamiento requeridos fueron desarrollados y aprobados.

- ¿El plan de prueba de aceptación es aprobado por el dueño del sistema, el usuario, y otros que financian el proyecto antes de conducir cualquier prueba de aceptación?
- ¿Las copias de los documentos del sistema se han proporcionado al personal de ayuda?

- ¿La lista de comprobación de la seguridad ha sido terminada por el dueño del sistema y remitida al encargado del plan de aseguramiento de calidad?

Aceptación. [27]

La aceptación del sistema de software se caracteriza generalmente como un proceso para aceptar oficialmente el software nuevo o modificado.

- ¿Se ha conducido el entrenamiento del usuario?
- Si existe un sistema actual, ¿la conversión del sistema y datos se ha realizado de acuerdo con el plan de conversión?
- En cada sitio que se ha instalado, ¿la facilidad se ha examinado para asegurar que la preparación de la instalación es completa y de acuerdo con el plan de instalación?

Dirección y Gestión de Configuración de Software. [27][28]

Configuración de Software generalmente se caracteriza como el control de cambios al software (incluso en la documentación) durante la iniciación, el desarrollo, y las fases de transiciones del ciclo de vida.

- ¿Se desarrollo un plan de configuración de software para el proyecto del software según un procedimiento estándar (es decir, ANSI/IEEE Std. 1042, etc.)?
- ¿Son tratados por la Gestión de Configuración de Software la identificación de la configuración, el control, la revisión, y la contabilidad del estado?
- ¿Existe un grupo responsable de establecer la Gestión de Configuración de Software en el proyecto?

Seguimiento de Proyecto. [27][30][31]

El seguimiento de proyecto de software generalmente se caracteriza como un proceso para establecer un estado de actividad contra lo planeado. Con esto se tiene un mecanismo para identificar problemas en el proceso, corregir deficiencias de proceso, y prevenir su repetición

- ¿La gerencia tiene un mecanismo para la revisión regular del estado de los proyectos del software?

- ¿Se pone al día el plan del proyecto cada vez que el estado del proyecto se desvía del plan?
- ¿Se utiliza un mecanismo para los cambios que controlan a los requisitos del software?

Planificación del Proyecto. [27] [29] [28] [31] [30]

El plan del proyecto del software se caracteriza generalmente como el proceso para seleccionar las estrategias, las políticas, los programas, y los procedimientos para alcanzar los objetivos y las metas del proyecto.

- ¿Se utilizan las estimaciones para el planeamiento del proyecto?
- ¿Se documentan las estimaciones (tamaño, coste, y horario) para el uso en el plan del proyecto?
- ¿se describe los antecedentes y el contexto del proyecto?

Aseguramiento de Calidad. [27] [28] [29] [30]

El Aseguramiento de calidad de software (SQA) es el proceso que mejora la calidad de productos del software, fortalece la satisfacción del cliente, y les proporciona la visibilidad apropiada a gerentes en el proceso del proyecto y productos. El proceso SQA asegura que un proyecto esté siguiendo sus prácticas y procesos. Las actividades de SQA son realizadas más fácilmente por un equipo especializado.

- ¿Hay alguna comprobación de que los productos y las actividades se adhieren a las normas del proyecto, procedimientos, y requisitos establecidos?
- ¿El equipo sigue las políticas escritas o la guía para llevar a cabo las actividades de SQA?
- ¿Se documentan los resultados de las revisiones?

Ingeniería de Software (Componentes) [29] [28]

El Software Diseñado basado en componentes está volviéndose común para el desarrollo de software de aplicación. Cuando el software usa este paradigma, se exige una considerable calidad de los componentes reusables que serán integrados en un nueva aplicación. La lista

de comprobación que sigue a continuación, puede ayudar a evaluar algunos de los problemas asociados con los componentes reusables.

- ¿La documentación del plan existente describe la interfaz para todos los componentes en la arquitectura del software?
- ¿Se han identificado para cada componente reusable en la interfaz y su funcionalidad?
- ¿Los componentes encajan dentro del estilo arquitectónico seleccionado?

Reingeniería. [29]

La lista de control siguiente puede ayudar a determinar qué aplicaciones son candidatas para una reingeniería. Las preguntas que reciben un positivo ("sí"), tienen una alta probabilidad para la reingeniería.

- ¿La aplicación tiene más de cinco años?
- ¿El valor anual de uso se conoce?
- ¿Es de uso crítico para el negocio?

Construcción de Equipo. [29]

Esta lista de comprobación es para determinar el grado, en el cual un equipo de proyecto del software trabajará con eficacia. Debe observar muchos factores que afectan a la calidad del un equipo.

- ¿Trabajaron los miembros en el equipo antes?
- ¿Los miembros del equipo tienen la mezcla correcta del conocimiento, del entrenamiento, y de la experiencia para que el trabajo sea hecho?
- ¿Los miembros del equipo vienen de la misma "cultura técnica" y de la misma organización?

Coordinación interna de la Organización. [28]

La coordinación permite que el proyecto satisfaga mejor las necesidades de cliente con eficacia y eficiencia, y ayuda al proyecto, a los requisitos, a los objetivos, y a las ediciones, en un nivel dirección del sistema. Los representantes de las organizaciones múltiples que

ganaron el proyecto participan en establecer los requisitos, los objetivos, y los planes a nivel sistema trabajando con el cliente y los usuarios finales, como es apropiado.

- ¿Todos los que participan en el proyecto aceptan los requisitos del cliente?
- ¿Todos los que participan en el proyecto aceptan los compromisos hechos entre las organizaciones del proyecto?
- ¿El proyecto identifica, rastrea, y soluciona los problemas internos?

Plan de Control de Cambio. [30]

Esta lista de comprobación cubre problemas relacionados con la creación del proceso de control cambio. El control de cambio es un subconjunto de la gestión de cambio que es un subconjunto de la gestión de configuración.

- ¿Los procesos de control de cambio son consistentes con los procesos del proyecto?
- ¿El plan de control de cambio es consistente con el plan de Gestión de Cambio?
- ¿El plan proyecto evita duplicación de información con el plan de Gestión de Configuración?

Riesgo del Proyecto. [29] [28]

Las prácticas de la gerencia de riesgo ayudan a identificar, analizar y controlar las áreas o los acontecimientos (problemas) del proyecto que pueden potencialmente afectar perjudicialmente al proyecto, (el horario se retrasa, costes, la baja calidad creciente.)

- ¿Se adaptan las actividades de gerencia de riesgo planeadas al proyecto?
- ¿El proyecto utiliza el(los) plan(es) de gerencia de riesgo para los riesgos significativos?
- ¿El proyecto identifica, analiza y controla riesgos significativos?

2.5.2 CICLO DE VIDA

Interfaz de Usuario.

Interfaz Gráfica de Usuario. [33] [34] [35] [29] [28] [30]

Los usos modernos del software han complicado a menudo las interfaces utilizadas. Porque el número de las líneas del código (o de componentes reutilizables)

requeridas para la puesta en práctica de la GUI (Graphic User Interface) puede exceder a menudo el número de líneas del código para otros elementos del software, la prueba cuidadosa de la interfaz utilizada es esencial.

- ¿Se asegura que cada caja de window/dialog tiene el valor prefijado claramente marcado (botón de comando, u otro objeto) y se invoca cuando se presiona la tecla caliente?
- ¿Se asegura que el foco esté fijado a un objeto?
- ¿Se asegura que los nombres del botón de opción (botón de radio) no sean abreviaturas?

Interfaz WEB. [36] [37] [29] [38] [39]

Sin importar que se utilicen técnicas de la ingeniería de Web para construir Web, es esencial determinar la calidad de la Web que se construyó. La lista de comprobación siguiente proporciona un conjunto de preguntas fundamentales que ayudan a determinar la calidad de una Web.

- ¿Qué nivel de experiencia tienen los usuarios con sistemas informáticos?
- ¿Y con la Web?
- ¿Y con esta interfaz y otras interfaces similares?

Determinación de Requisitos. [27] [28] [40] [30][34] [48] [49]

La práctica de la gerencia de los requisitos establece y mantiene una comprensión común con el cliente y el proyecto con respecto a los requisitos del software. Este acuerdo forma la base para estimar, planear, realizar, y seguir actividades del proyecto a través del ciclo de vida del software.

- ¿Los requisitos se escriben en lengua comprensible no técnica?
- ¿Todos los requisitos relacionados con el funcionamiento se incluyen?
- ¿Hay requisitos que describen al mismo objeto que están en conflicto con el resto de las características?

Determinación de Requisitos no Funcionales. [30]

Los requisitos cubren el descubrimiento, la documentación, y el análisis de las funciones que se pondrán en ejecución en software. El análisis de requisitos es una actividad crítica en

cualquier proyecto del software porque si usted no consigue los requisitos derechos, su proyecto fallará no importa cómo esta' bien usted realiza el resto de las actividades.

- ¿Se indican los requisitos en una forma específica y mensurable?
- ¿Los requisitos de la confiabilidad han sido especificados?
- ¿Los requisitos de la disponibilidad han sido especificados?
- ¿Los requisitos de la funcionalidad han sido especificados?
- ¿Los requisitos de la robustez han sido especificados?

Análisis de Software. [29] [48] [49]

El análisis es la creación de todas las actividades de la ingeniería de software subsiguiente.

- ¿Todas las funciones han sido descompuestas?
- ¿Las especificaciones del proceso se han desarrollado para cada transformación de nivel bajo?
- ¿Una herramienta CASE se ha usado para apoyar a los modelos del análisis?

Diseño de Software. [27] [29] [30] [48] [49]

El diseño de software se inicia con un proceso de traducción de la documentación del sistema de requisitos dentro del diseño funcional orientado al usuario. El dueño del sistema, usuario y equipo de proyecto finaliza este diseño y usa este como base para un diseño de sistema más técnico.

- ¿Un documento de diseño de sistema es creado?
- ¿Un diseño de sistema es aceptado y encaminado?
- ¿La notación del diseño es estandarizada?

Experimentación. [27][29][28] [49]

La experimentación se caracteriza generalmente como la validación de los requisitos. Se han sido cubiertos los requisitos o están en un nivel aceptable de acuerdo con los estándares definidos al inicio del desarrollo o por los atributos de calidad definidos.

Entrenamiento. [27] [28] [30]

Los objetivos del entrenamiento se resumen como sigue:

- Asegurar a todo el personal asignado a utilizar el sistema de acuerdo a su función y responsabilidad.
- Asegurar que la documentación del sistema refleja exactamente al sistema.
- Asegurar la estructura para el entrenamiento futuro.
    - ¿Los vehículos del entrenamiento para impartir habilidades y conocimiento se identifican y se aprueban?
    - ¿Las herramientas para apoyar las actividades del programa de entrenamiento están disponibles?
    - ¿Las medidas se hacen y se utilizan para determinar el estado de las actividades del programa de entrenamiento?

Validación. [29]

Durante la validación, el software se prueba contra los requisitos. Esta lista de comprobación trata de algunas de las cuestiones claves que conducirán a la validación eficaz.

- ¿Existe una especificación de requisitos completa del software disponible?
- ¿Se limitan los requisitos?
- ¿Las clases se han definido para ejercitar la entrada?

Arquitectura. [30] [49]

Esta lista de comprobación cubre diseños en el nivel de la arquitectura. El término elemento se utiliza en este documento para significar cualquier subsistema, componente, módulo, u otro elemento del diseño que la arquitectura cubre.

- ¿La arquitectura comunica una visión adecuada del sistema que las actividades de diseño posteriores enfocan?
- ¿La arquitectura se diseña para acomodar cambios probables?
- ¿La arquitectura se basa en actividades del diseño detallado de componentes y de la interfaz?

Código General. [30]

Esta lista de comprobación cubre las ediciones de codificación que se relacionan con cualquier tipo de actividad.

- ¿Usted entiende el diseño que está a punto de construir?
- ¿El diseño proporciona un nivel apropiado del detalle para comenzar?
- ¿Es factible construir el código apartir del diseño de forma directa o debe él ser revisado antes de procurar construirlo?

Módulos de Código. [30]

El término módulo se utiliza para describir una unidad coherente del código de fuente que tiene un propósito específico, por ejemplo, una clase en lenguaje Orientado a Objetos. Los módulos se contienen normalmente en archivos, pero se pueden almacenar en otras maneras dependiendo de la tecnología que el módulo desarrolla.

- ¿El módulo tiene un propósito central?
- ¿El módulo es definido en el diseño de bajo nivel del sistema?
- ¿El módulo ofrece un sistema cohesivo de servicios?

Código de Programa Orientada a Objetos (OOP). [30]

Esta lista de comprobación cubre las ediciones hechas por los programadores de codificación específicas para la programación orientada objetos (OOP) que no están cubiertos en las otras listas de comprobación.

- ¿El lenguaje y las tecnologías seleccionadas para realizar un diseño orientado objeto apoyan al diseño OO?
- ¿Hay una comprensión clara de cómo el lenguaje y la tecnología seleccionada pueden afectar aspectos de cómo el modelo del objeto será construido, especialmente respecto a la creación, comunicación, y persistencia?
- ¿Hay una comprensión clara de cómo los aspectos estáticos y dinámicos del modelo del objeto afectarán la construcción, especialmente respecto a la herencia por clasificación, la asociación, la agregación, y la delegación?

2.5.3 ATRIBUTOS DE CALIDAD

Fiabilidad y Confiabilidad (Seguridad). [27]

La preparación de la aceptación del sistema de software se caracteriza generalmente por un proceso que asegura todo el lugar para comenzar las actividades del proceso de la aceptación. La lista de comprobación de seguridad ayuda a asegurarse de que se han terminado todas las actividades necesarias de la seguridad.

- ¿Las reglas de la protección de los datos campos, archivos, las autoridades, y los códigos apropiados de la identificación de usuario fueron establecidos por el dueño del sistema o según lo asignado por mandato o por una autoridad más alta?
- ¿La protección del control de acceso fue incorporada en el sistema?
- ¿Todo el aislamiento, la libertad de la información, la sensibilidad, y las consideraciones de la clasificación fueron identificadas, resueltas, y establecidas?

Integridad. [29]

Cuando se construye el software, es necesario considerar un número de versiones importantes. La lista de comprobación siguiente puede ser aplicada mientras que las pruebas de la integración se planean y durante el proceso mismo de la integración.

- ¿Se ha establecido un horario de la integración?
- ¿Se realiza la prueba de la regresión mientras que se integran los módulos nuevos?
- ¿Si se ha elegido una estrategia bottom-up, están los participantes disponibles para poder probar adecuadamente los módulos inferiores?

Facilidad de Uso. [41] [42] [43] [44] [36]

Todos quisieran que la aplicación sea tan fácil de usar como sea posible. A veces en el calor de la batalla del diseño, se olvida de seguir las pautas básicas de la utilidad. Aquí está una lista de los principios de la utilidad que pueden probar.

- ¿Se consideran otros idiomas para las instrucciones en la pantalla de forma adecuada?
- ¿Hay ausencia de términos en idiomas diferentes mezclados?

- ¿Es simple el vocabulario utilizado?

Mantenimiento de Software.

Dentro del mantenimiento se realiza la resolución de los errores, de los defectos, y de las fallas del software respecto a los requisitos, iniciando cambios en el ciclo de vida del software. El cambio se planea y se ejecuta, de tal modo el proceso del mantenimiento se convierte en iteraciones del desarrollo.

- ¿Existe un procedimiento para manejar los cambios de emergencia?
- ¿Un número de identificación se asigna a la modificación?
- ¿La modificación se clasifica como: correctiva, adaptable, emergencia, programable, perfectiva, obligatoria, requerida, o agradable?

Facilidad de Prueba. [28]

La ingeniería de software para el producto, describe una práctica que abarca el proceso entero de creación. Un producto de software es el sistema completo de artículos entregados a los clientes, y puede incluir software, documentación, o datos.

- ¿Las actividades, los métodos y las prácticas esenciales se planean para el desarrollo y/o el mantenimiento de los productos de software lanzados?
- ¿Las actividades se adaptan al proyecto?
- ¿El personal del proyecto recibe el entrenamiento necesario para sus tareas?

Interoperabilidad. [28] [30]

Los formatos estándares en las practicas de interoperabilidad de datos y base de datos se trasfiere entre sistemas de múltiple aplicaciones. Un dato puede ser compartido entre dos o mas aplicaciones, o quizá es llamado y usado internamente dentro de una sola aplicación.

- ¿Los estándares usados para los datos y base de datos están implementados?
- ¿La estructura de los datos y base de datos facilitan la interoperabilidad con otros sistemas?
- ¿Las actividades de la interoperabilidad de datos, base de datos y resultados periódicos son revisados con un administrador de proyecto?

2.6 CONCLUSIONES

En el modelo propuesto para las inspecciones, existen tres etapas generales, que son la Planificación, Verificación y Resultados. Estas tres etapas, para una empresa pequeña, se pueden convertir en Planificación, Verificación Sincrónica y Resumen de Defectos. Con estas tres etapas, se puede lograr realizar una inspección adecuada a proyectos relativamente pequeños. Es claro que, en una empresa pequeña no existirán todos los roles que participan en una inspección, sin embargo, el coordinador, un asesor y un inspector podrán realizar los demás roles.

En general, el nuevo modelo da una versatilidad y un control aceptable a los inspectores sobre el artefacto a inspeccionar, logrando, en una verificación asincrónica, detectar defectos diversos, ya que el inspector estará sin presión, ni manipulación y sin ser influenciado por el grupo. El inspector, realmente con la habilidad y conocimiento que posee será el que detecte el defecto.

Se ha detallado los distintos roles de participación en una inspección, estos son muy determinantes para el éxito, no deben ser interferidos, ni ser manipulados por otra persona que no sea el Coordinador, él es el que conoce las habilidades y capacidades de cada uno de los inspectores que participa en los proyectos, por esta razón él es el mas indicado para la planificación y selección del equipo de inspección. El Coordinador debe tener la habilidad innata de la administración y distribución de recursos.

También se observa, que las listas de comprobación juegan un papel muy importante al momento de detectar los defectos, la clasificación de las listas da una idea muy clara que los proyectos de software deben de ser muy bien controlados para no llegar al fracaso. Para, iniciar a una empresa de software en el camino de certificación nacional o internacional, las listas de comprobación pueden convertirse en reglas que el desarrollador siga para la construcción del software, de esta forma, a medida que pase el tiempo se puede seleccionar algunas de ellas para tener una norma interna para el desarrollo de software o realizar un listado de los posibles defectos en cada una de las etapas de desarrollo, para prevenir los errores que cometan los participantes en la construcción.

La parte más importante dentro de las inspecciones es la etapa de resumen de defectos, ya que en esta etapa se puede definir los cambios dentro del proceso definido en la empresa, teniendo como resultado una mejora en el proceso.

CAPITULO 3 DESCRIPCIÓN DEL SOFTWARE

3.1 INTRODUCCIÓN

En este último capitulo, se definen las características más importantes de la aplicación que automatiza el modelo propuesto para la inspección de software descrito en el anterior capítulo.

Primero se explica brevemente la plataforma donde la aplicación se ejecuta, además, se describe el hardware y software necesario para que el software desarrollado pueda ser ejecutado y sea mantenido. La descripción anterior está acompañada por una justificación del por qué se escogió las plataformas de hardware y software, para la automatización del modelo propuesto de inspección de software.

También, se especifica los Requisitos Funcionales y no Funcionales, el patrón cliente/servidor en sus distintas capas, una descripción general de los casos de uso, un diagrama de clases, un modelo de datos y para finalizar un análisis del costo/beneficio.

3.2 ESPECIFICACIÓN GENERAL DEL SOFTWARE

El software permite gestionar el Proceso de Inspección de Software, ayudando al personal de Aseguramiento de Calidad en las siguientes tareas: crear nuevas listas de comprobación para la verificación de los artefactos, planificar una nueva inspección, capturar los resultados de la evaluación, clasificar los defectos detectados por su gravedad, registrar las posibles soluciones de los defectos, generar un resumen de defectos y posibles soluciones, y ayudar a la preparación de los inspectores.

El software está desarrollado para un entorno Web, dando una portabilidad amplia para el usuario final, requiriendo tan solo un navegador Web y una conexión interna de red para la comunicación con el servidor Web. El servidor, debe cumplir con los requisitos de hardware mínimos que especifica el Microsoft Visual Studio .NET y las especificaciones de software requerida para el funcionamiento correcto del servidor Web. Los requisitos mínimos de hardware son los siguientes:

- Procesador: Pentium II 450 Mhz.

- Memoria: 96 Mb de memoria RAM.
- Disco Duro: 900 Mb de espacio disponible en la unidad del sistema. 3,3 Gb de espacio en la unidad de instalación.
- Unidad de Disco: Unidad de CD-ROM o DVD-ROM.
- Monitor: Resolución de Súper VGA (1024 x 768) o superior con 256 colores.
- Mouse: Microsoft Mouse o compatible.

El requisito de software mínimo para el funcionamiento de Visual Studio .NET es el siguiente:

- Sistema Operativo: Windows 2000 Professional.

También es necesaria la instalación de Microsoft SQL Server 2000, ya que el software desarrollado interactúa con una base de datos relacional. Los requisitos mínimos del sistema son los siguientes:

- Sistema Operativo: Windows NT Server 4.0, Enterprise Edition, con SP5 o posterior.
- Memoria: 64 Mb memoria RAM.
- Disco Duro: 270 Mb.
- Unidad de Disco: Unidad CD-ROM .
- Monitor: VGA o SVGA.
- Otros Dispositivos: Microsoft Internet Explorer version 5.0 o superior.

3.3 JUSTIFICACIÓN DE LOS MÉTODOS Y HERRAMIENTAS

3.3.1 MÉTODOS Y HERRAMIENTAS

La aplicación esta desarrollada en una plataforma Windows, ya que el Visual Studio .NET no existe para otra plataforma, pero corresponde con el medio mas utilizado de desarrollo en el país. Las ventajas que ofrece la Micro Arquitectura .NET para el desarrollo de aplicaciones son las siguientes [65]:

- Visual Studio .NET 2003 es una completa herramienta para crear con rapidez aplicaciones conectadas a Microsoft .NET para Microsoft Windows y Web, que aumenta notablemente la productividad de los programadores y permite nuevas oportunidades de negocio y empresariales.

- Comparte en forma eficaz y segura el código fuente, documentos de diseño y otros activos de desarrollo con funcionalidad de control de versiones integrada. Aprovecha el Lenguaje de descripción de plantillas basado en XML y los proyectos de Enterprise Templates para compartir directrices de desarrollo y procedimientos recomendados entre los equipos de desarrollo.
- Puede crearse y ejecutarse pruebas con rapidez para optimizar el rendimiento y la escalabilidad de aplicaciones y servicios Web XML utilizando Application Center Test.
- Puede crearse con rapidez aplicaciones y servicios Web XML que abarquen numerosos dispositivos y plataformas, y elija entre una gran variedad de lenguajes de programación.
- Puede crearse aplicaciones escalables de alto rendimiento utilizando versiones completas para programadores de servidores, incluidos Microsoft Windows Server 2003, Microsoft SQL Server™, Microsoft Exchange Server, Microsoft Commerce Server y Microsoft Host Integration Server.
- Se puede elegir entre un gran variedad de herramientas de ciclo de vida y productividad incorporadas que se integran en el entorno de Visual Studio .NET para afrontar todas las etapas del ciclo de desarrollo de software. Se puede seleccionar el mejor lenguaje para la necesidad de desarrollo de una aplicación entre mas de veinte lenguajes compatibles con Windows .Net Framework.

La aplicación esta desarrollada en ASP.NET, que es un marco de trabajo de programación generado en Common Language Runtime que puede utilizarse en un servidor para generar eficaces aplicaciones Web, es una extensión del Visual Studio .NET. ASP.NET ofrece varias ventajas importantes acerca de los modelos de programación Web anteriores:

- **Mejor rendimiento.** ASP.NET es un código de Common Language Runtime compilado que se ejecuta en el servidor. A diferencia de sus predecesores, ASP.NET puede aprovechar las ventajas del enlace anticipado, la compilación just-in-time, la optimización nativa y los servicios de caché desde el primer momento. Esto supone un incremento espectacular del rendimiento antes de siquiera escribir una línea de código.

- **Compatibilidad con herramientas de primer nivel.** El marco de trabajo de ASP.NET se complementa con un diseñador y una caja de herramientas muy completos en el entorno integrado de programación (Integrated Development Environment, IDE) de Visual Studio.

- **Eficacia y flexibilidad.** Debido a que ASP.NET se basa en Common Language Runtime, la eficacia y la flexibilidad de toda esa plataforma se encuentra disponible para los programadores de aplicaciones Web. La biblioteca de clases de .NET Framework, la Mensajería y las soluciones de Acceso a datos se encuentran accesibles desde el Web de manera uniforme. ASP.NET es también independiente del lenguaje, por lo que puede elegir el lenguaje que mejor se adapte a la aplicación o dividir la aplicación en varios lenguajes. Además, la interoperabilidad de Common Language Runtime garantiza que la inversión existente en programación basada en COM se conserva al migrar a ASP.NET.

- **Simplicidad.** ASP.NET facilita la realización de tareas comunes, desde el sencillo envío de formularios y la autenticación del cliente hasta la implementación y la configuración de sitios. Por ejemplo, el marco de trabajo de páginas de ASP.NET permite generar interfaces de usuario, que separan claramente la lógica de aplicación del código de presentación, y controlar eventos en un sencillo modelo de procesamiento de formularios de tipo Visual Basic. Además, Common Language Runtime simplifica la programación, con servicios de código administrado como el recuento de referencia automático y el recolector de elementos no utilizados.

- **Facilidad de uso.** ASP.NET emplea un sistema de configuración jerárquico, basado en texto, que simplifica la aplicación de la configuración al entorno de servidor y las aplicaciones Web. Debido a que la información de configuración se almacena como texto sin formato, se puede aplicar la nueva configuración sin la ayuda de herramientas de administración local. Esta filosofía de "administración local cero" se extiende asimismo a la implementación de las aplicaciones ASP.NET Framework. Una aplicación ASP.NET Framework se implementa en un servidor sencillamente mediante la copia de los archivos necesarios al servidor. No se requiere el reinicio

del servidor, ni siquiera para implementar o reemplazar el código compilado en ejecución.

- **Escalabilidad y disponibilidad.** ASP.NET se ha diseñado teniendo en cuenta la escalabilidad, con características diseñadas específicamente a medida, con el fin de mejorar el rendimiento en entornos agrupados y de múltiples procesadores. Además, el motor de tiempo de ejecución de ASP.NET controla y administra los procesos de cerca, por lo que si uno no se comporta adecuadamente (filtraciones, bloqueos), se puede crear un proceso nuevo en su lugar, lo que ayuda a mantener la aplicación disponible constantemente para controlar solicitudes.

- **Posibilidad de personalización y extensibilidad.** ASP.NET presenta una arquitectura bien diseñada que permite a los programadores insertar su código en el nivel adecuado. De hecho, es posible extender o reemplazar cualquier subcomponente del motor de tiempo de ejecución de ASP.NET con su propio componente escrito personalizado. La implementación de la autenticación personalizada o de los servicios de estado nunca ha sido más fácil.

- **Seguridad.** Con la autenticación de Windows integrada y la configuración por aplicación, se puede tener la completa seguridad de que las aplicaciones están a salvo.

3.4 REQUISITOS DEL SOFTWARE

3.4.1 REQUISITOS FUNCIONALES

1. Capturar Lista de Comprobación Estándar.

1.1 Capturar datos de la lista, las cuales contienen: Nombre, Propósitos y Descripción.

1.2 Una lista consta de uno o más objetivos.

1.3 Capturar los objetivos de la lista los cuales contienen: Nombre, Descripción y Peso.

1.4 El Peso de un objetivo es la medida de su importancia, el cual varía de 0 a 10.

1.5 Para cada objetivo se tienen muchas preguntas y una misma pregunta se encuentra en distintos objetivos.

1.6 Capturar las preguntas las cuales contienen: Nombre, Descripción y Peso.

1.7 El Peso de una pregunta es la medida de su importancia, el cual varía de 0 a 10.

2 Capturar un Inspector.

2.1 Capturar los datos del inspector los cuales contienen: Nombre, Área, Correo, Estado (disponible o no) y especialidad.

2.2 Un inspector puede tener (1) o mas especialidades.

2.3 Una especialidad consta de un Nombre y de una descripción.

3 Capturar un Proyecto y clasificarlo.

3.1 Capturar el nombre y propósito, ubicación y formato.

3.2 Un proyecto puede ser clasificado en: Software del sistema, Software de tiempo real, Software de gestión, Software de ingeniería y científico, Software embebido, Software de computadoras personales, Software de inteligencia artificial y Software educativos.

4 Crear una lista a partir de listas estándares.

4.1 Capturar los datos de la nueva lista, los cuales contienen: Nombre, Propósito y Descripción.

4.2 Establecer los objetivos y preguntas de la nueva lista.

4.2.1 Seleccionar los objetivos y preguntas definidos en una lista estándar

4.2.2 Crear Objetivos y/o preguntas

4.3 Asociar los objetivos y las preguntas definidos con la nueva lista.

5 Planificar las Inspecciones.

5.1 Capturar los datos de la inspección los cuales constan de: Nombre del artefacto, propósito, fecha de inicio, fecha de finalización, ubicación física del artefacto, desarrollador y etapa que pertenece el artefacto.

5.2 Por cada proyecto a inspeccionar capturar los datos de la agenda, los cuales contienen: fecha de inicio, fecha de finalización, hora y lugar.

5.3 Por cada inspección a un proyecto elegir a los inspectores que participaran. El número de los inspectores debe ser mayor a dos y menos de cinco.

5.3.1 Por los atributos y especialidad se asigna un Jefe de inspección de los dos o cinco seleccionados y esta asignación la lleva a cabo el jefe de SQA o coordinador.

5.3.2 Seleccionar a un Asesor, el cual ayudará a la planificación de la inspección el cual es seleccionado por el jefe de inspección o coordinador.

5.3.3 Asignar un Apuntador que controlará la buena semántica del por qué de los defectos encontrados y este es seleccionado por el jefe de Inspección juntamente con el Asesor.

5.4 Capturar los documentos de apoyo necesarios para la inspección, los cuales son necesarios para mantener informado al inspector.

5.4.1 Establecer la ubicación física y el formato de cada uno de los documentos de apoyo, para que el inspector pueda utilizar, ya sea en su preparación o en la verificación.

5.4.2 Capturar documentos adicionales, para que el inspector tenga conocimiento y se prepare adecuadamente para encontrar los defectos.

5.5 Capturar el propósito de la inspección.

5.6 Asignar las Listas de Comprobación que se utilizarán en la inspección.

5.7 Generar los defectos posibles a encontrar (por inspecciones anteriores).

6 Capturar a los Desarrolladores, responsables de los artefactos a inspeccionar.

6.1 Capturar los datos de los desarrolladores, los cuales son: Nombre desarrollador, descripción.

7 Generar paquete de inspección.

7.1 El paquete de inspección está constituido por:

7.1.1 Inspectores que participarán en la inspección.

7.1.2 Listas de comprobación a usar en la inspección.

7.1.3 Relación entre Inspectores-Listas

7.1.4 Documentos de apoyo a la inspección.

8 Capturar resultados de las inspecciones

8.1 Capturar los defectos encontrados a través de la lista de comprobación por cada una de sus preguntas, especificados con MUY GRAVE, GRAVE, ACEPTABLE, LEVE y las observaciones a las preguntas que tengan esta escala de defecto.

8.2 Capturar el tiempo empleado por cada Inspección.

8.3 Capturar el tiempo empleado por cada Inspector.

8.4 Capturar el número de defectos encontrados por cada inspección.

8.5 Capturar el número de defectos encontrados por cada inspector.

9 Generar informe final.

9.1 General un resumen de defectos encontrados mas las posibles soluciones a los defectos.

10 Capturar los datos de las correcciones o modificaciones.

10.1 Capturar por cada pregunta con defecto las posibles soluciones y modificaciones que se pueden realizar para su corrección.

10.2 Capturar las fechas modificación del defecto corregido.

3.4.2 DEFINICIÓN DE ATRIBUTOS NO FUNCIONALES.

Los atributos o requisitos no funcionales son las propiedades o cualidades que el producto de software debe tener, tienen que ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento, interfaces de usuario, mantenimiento, seguridad, portabilidad, estándares, etc.

A continuación se enuncian los requisitos no funcionales más importantes del sistema propuesto:

Requisitos de apariencia o interfaz externa

El sistema debe ser fácil de usar y su uso debe ser más una ayuda al trabajo de los usuarios que un peso adicional.

Requisitos de la Facilidad de Uso

Debido a que este software está destinado a ser usado por personas relativamente familiarizadas con la informática se consideran aceptables o/y necesarias en el producto, las siguientes características:

- Un aprendizaje sencillo (para informáticos).
- Proporcione un incremento de la capacidad de detección de los defectos en cada uno de los proyectos.

- El lenguaje por defecto es el castellano.
- Debe presentar consistencia y claridad en la interfaz de usuario.
- No debe presentar abreviaturas en las palabras utilizadas en la interfaz.

Requisitos de Eficiencia y Eficacia

En general el sistema debe ser rápido y responder en un tiempo adecuado como para clasificarlo como interactivo. Una excepción a esta regla puede ser en el caso de la generación de reportes, los cuales dependen de la cantidad de datos a analizar y su tiempo de respuesta puede no ser tan rápido. Aunque la rapidez es un punto deseado en todo sistema, con vistas a aumentar la satisfacción del usuario, en este caso, la velocidad de operación no es un punto crítico.

Requisitos de Software

El componente servidor de este sistema requiere para su funcionamiento del servidor de base de datos MS SQL–Server 2000 o superior por tanto necesita del sistema operativo MS Windows y un servidor de páginas Web compatible con .NET. Para su ejecución necesita además de la tecnología .NET (Framework, CLR - Common Language Runtime, etc.) la cual hasta el momento solo esta disponibles para Windows y Linux. En las máquinas de los usuarios bastará con cumplir con los requisitos de un navegador Web (recomendado MS Internet Explorer o Netscape).

Requisitos de Portabilidad

La disponibilidad de este producto para distintas plataformas esta condicionada por la existencia del CLR (Common Language Runtime) y jerarquía de clases del .NET para distintos sistemas operativos, al momento de desarrollo del sistema se conoce de la existencia de dichos componentes solamente para Linux y Windows. De ahí que la implementación del software para otras plataformas como Linux se basa en la viabilidad de conversión de la base de datos hacia un gestor compatible y de la salida al mercado de versiones del Framework de .NET para la plataforma deseada. Sin embargo, para la aplicación de los usuarios finales, solo el hecho de que la tecnología ASP.NET se basa en las paginas WEB y el protocolo de comunicación HTML, se puede asegurar que este

requisito se cumple para muchas plataformas (Windows, Mac, Linix, Unix), basta con un interprete HTML.

Requisitos documentación en línea

Todos los reportes, pantallas de entrada de datos, métricas y gráficos que se obtengan deben incluir información adicional a manera de ayuda. Cada componente visual deberá contener una explicación completa de todos sus elementos y una interpretación de los posibles valores.

Requisitos de Seguridad

Confidencialidad: La información manejada por el sistema estará protegida contra accesos no autorizados y la información crítica como claves de usuarios, etc. será correctamente encriptada en el sistema y manejada por .NET.

Integridad: La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes, tanto a nivel de aplicación como en el servidor de base de datos.

Disponibilidad: A los usuarios autorizados se les garantizará el acceso a la información y que los dispositivos o mecanismos utilizados para lograr la seguridad no ocultarán o retrasarán el acceso a los datos deseados en un momento dado.

3.5 CAPAS DEL DISEÑO DE LA APLICACION

La aplicación desarrollada utiliza una arquitectura cliente/servidor con las siguientes capas:

- Capa de presentación de datos.
- Capa del proceso de la interfaz.
- Capa de la lógica de la aplicación.
- Capa de almacenamiento.

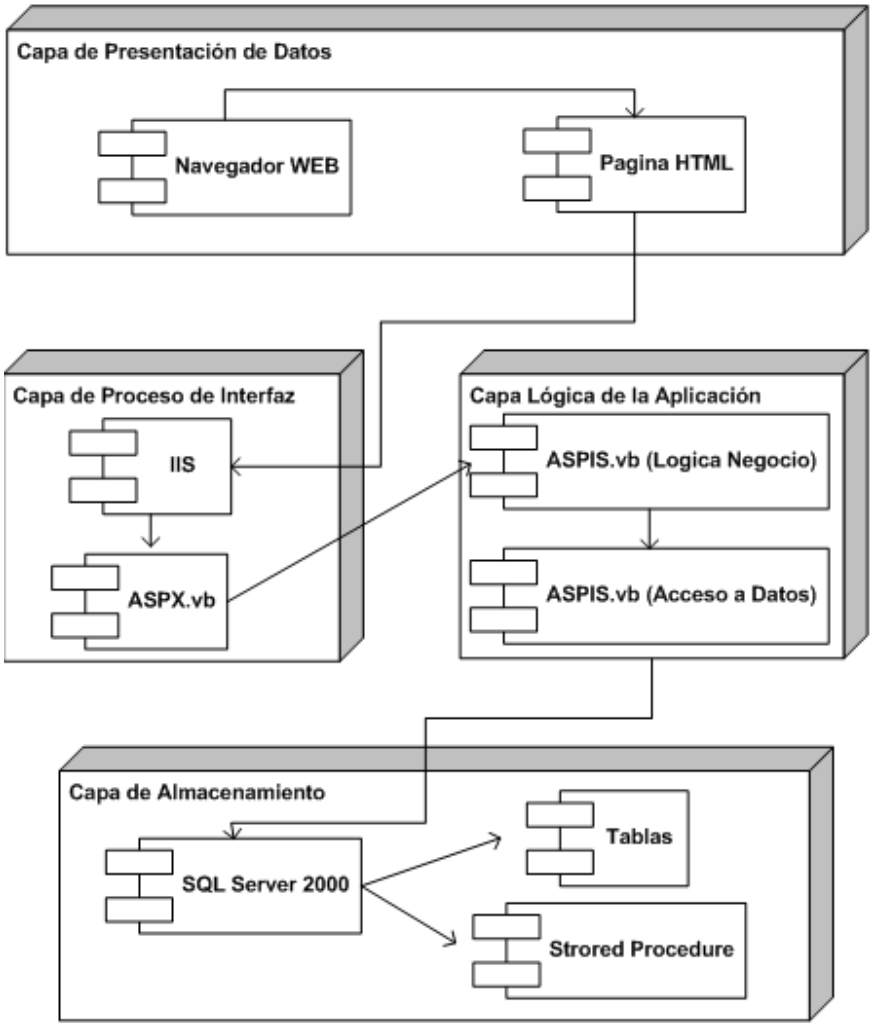


Figura 3.1 Capas de la Aplicación SPIS

Como se observar en la Figura 3.1, el comportamiento para el cliente será muy ligero (Cliente Delgado), ya que el Navegador WEB solo se encargará de mostrar y capturar los datos requeridos por el usuario, no realizará ninguna operación adicional lógica de la aplicación, ni del negocio. El Internet Information Server (IIS) será el encargado de administrar y controlar las páginas que se mostrarán en el Navegador WEB con una lógica descrita de código en los formularios ASPX.VB. Estos formularios tendrán una interacción con el fichero ASPIS.VB, en el cual se implementa la lógica del negocio y el acceso a los datos. Los datos almacenados por el cliente estarán en el Microsoft SQL – Server 2000, manejando tablas especificas del modelo relacional y los procedimientos almacenados que ayudan a la manipulación de los datos.

Para mayor detalle sobre la arquitectura de la aplicación vea el Anexo D.

3.6 CASOS DE USO

3.6.1 DEFINICIÓN DE LOS CASOS DE USO

CASO DE USO	Capturar lista de comprobación
ACTORES	Coordinador
DEFINICIÓN	El Coordinador será el que capte todos los datos de las listas estándares establecidos o creen nuevas listas de comprobación para una inspección.
REFERENCIA	RF.1, RF. 4

CASO DE USO	Asignar Objetivos a la Lista de Comprobación
ACTORES	Coordinador
DEFINICIÓN	El Coordinador seleccionará los Objetivos que considere importantes en ese momento para la lista de comprobación.
REFERENCIA	RF.1.2, RF.1.3, RF 1.4, FR 1.5

CASO DE USO	Asignar Preguntas a los Objetivos
ACTORES	Coordinador
DEFINICIÓN	El Coordinador seleccionará las preguntas que contendrá cada objetivo ya seleccionado.
REFERENCIA	RF. 1.6, RF. 1.7

CASO DE USO	Planificar Inspección
ACTORES	Jefe de SQA, Planificador.
DEFINICIÓN	El jefe de SQA seleccionará un Jefe de Inspección y este a su vez un Asesor y estos últimos llevarán a cabo la planificación de toda la inspección.
REFERENCIA	RF.2, RF.3, RF.5, RF.6

CASO DE USO	Preparar Inspector
ACTORES	Jefe de SQA, Jefe de inspección, Inspectores
DEFINICIÓN	Los distintos inspectores realizarán una petición al Jefe de Inspección para tener en su poder los documentos que conforman y ayudarán a la inspección, supervisados por el Jefe de SQA.
REFERENCIA	RF.7

CASO DE USO	Capturar resultado de la inspecciones
ACTORES	Jefe de inspección, Inspectores, Apuntador
DEFINICIÓN	El grupo de los inspectores, coordinado por el Jefe de

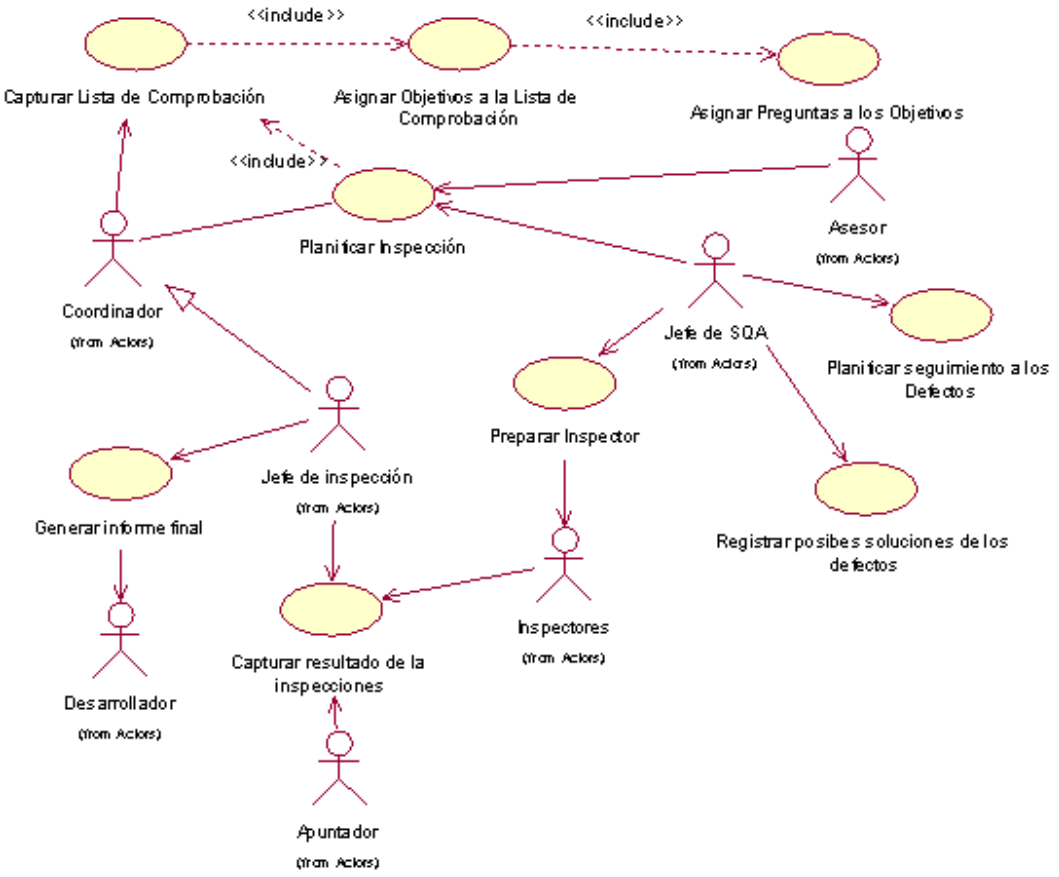
	inspección realizarán la detección de los defectos ayudados por el apuntador que será el encargado de la semántica del por qué de los defectos
REFERENCIA	RF.8

CASO DE USO	Generar informe final
ACTORES	Apuntador, Jefe de Inspección
DEFINICIÓN	El Apuntador dirigido por el Jefe de inspección y al frente de los demás integrantes Asesor y Desarrollador resumirá los resultados de la inspección.
REFERENCIA	RF. 9

CASO DE USO	Planificar seguimiento de los defectos
ACTORES	Desarrollador, Jefe de Inspección
DEFINICIÓN	El jefe de inspección juntamente con el desarrollador planificará una fecha tope para la corrección de los defectos encontrados en la inspección.
REFERENCIA	RF. 10

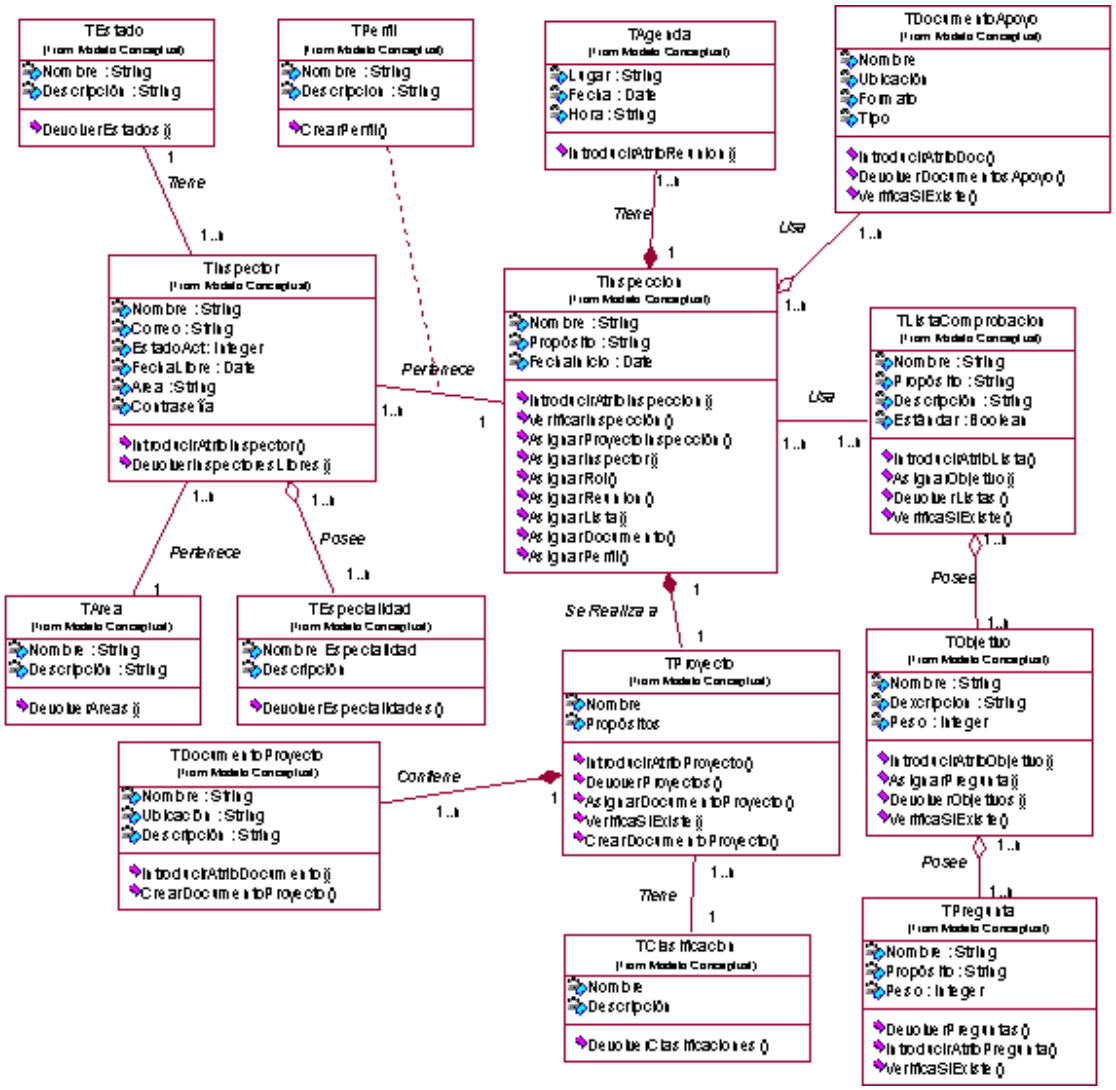
CASO DE USO	Registrar Posibles Soluciones
ACTORES	Jefe de Inspección
DEFINICIÓN	El jefe de inspección juntamente con el Asesor registrará las posibles soluciones de reconstrucción para los defectos encontrados en la inspección realizada.
REFERENCIA	RF. 10.1

3.6.2 DIAGRAMA DE CASOS DE USO



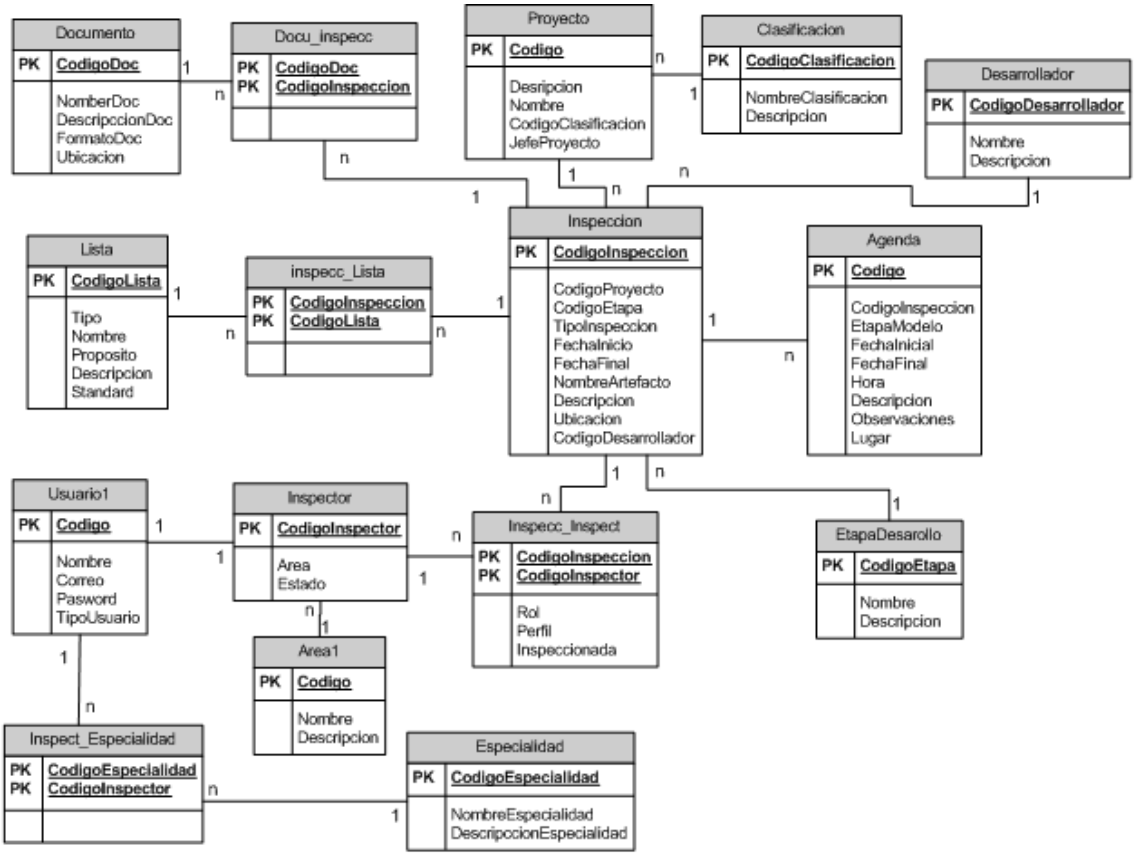
3.7 DIAGRAMA DE CLASES

El diagrama del diseño Web del sistema se obtiene como resultado del refinamiento del modelo conceptual [25], se basa en los mapas de navegación y en los diagramas de interacción (ANEXO C). A continuación se presenta el diagrama de clases.



3.8 MODELO DE DATOS

Del análisis del diagrama de clases para la definición de las clases persistentes y atributos [66] [67], se realizo el modelo de datos, que representa la base de datos para la aplicación SPIS, la cual está implementada en un administrador de base de datos relacional que es el Microsoft SQL-Server 2000. Para una mayor comprensión del modelo, se ha dividido en dos partes, la primera representa la etapa de Planificación del modelo propuesto y la segunda a las etapas de Verificación y Resultados.


Documento
PK CodigoDoc
NomberDoc
DescripccionDoc
FormatoDoc
Ubicacion
Docu_inspecc
PK CodigoDoc
PK CodigoInspeccion
Proyecto
PK Codigo
Desripcion
Nombre
CodigoClasificacion
JefeProyecto
Clasificacion
PK CodigoClasificacion
NombreClasificacion
Descripcion
Desarrollador
PK CodigoDesarrollador
Nombre
Descripcion
Lista
PK CodigoLista
Tipo
Nombre
Proposito
Descripcion
Standard
inspecc_Lista
PK CodigoInspeccion
PK CodigoLista
Inspeccion
PK CodigoInspeccion
CodigoProyecto
CodigoEtapa
TipoInspeccion
FechaInicio
FechaFinal
NombreArtefacto
Descripcion
Ubicacion
CodigoDesarrollador
Agenda
PK Codigo
CodigoInspeccion
EtapaModelo
FechaInicial
FechaFinal
Hora
Descripcion
Observaciones
Lugar
Usuario1
PK Codigo
Nombre
Correo
Pasword
TipoUsuario
Inspector
PK CodigoInspector
Area
Estado
Inspecc_Inspect
PK CodigoInspeccion
PK CodigoInspector
Rol
Perfil
Inspeccionada
EtapaDesarollo
PK CodigoEtapa
Nombre
Descripcion
Area1
PK Codigo
Nombre
Descripcion
Inspect_Especialidad
PK CodigoEspecialidad
PK CodigoInspector
Especialidad
PK CodigoEspecialidad
NombreEspecialidad
DescripccionEspecialidad

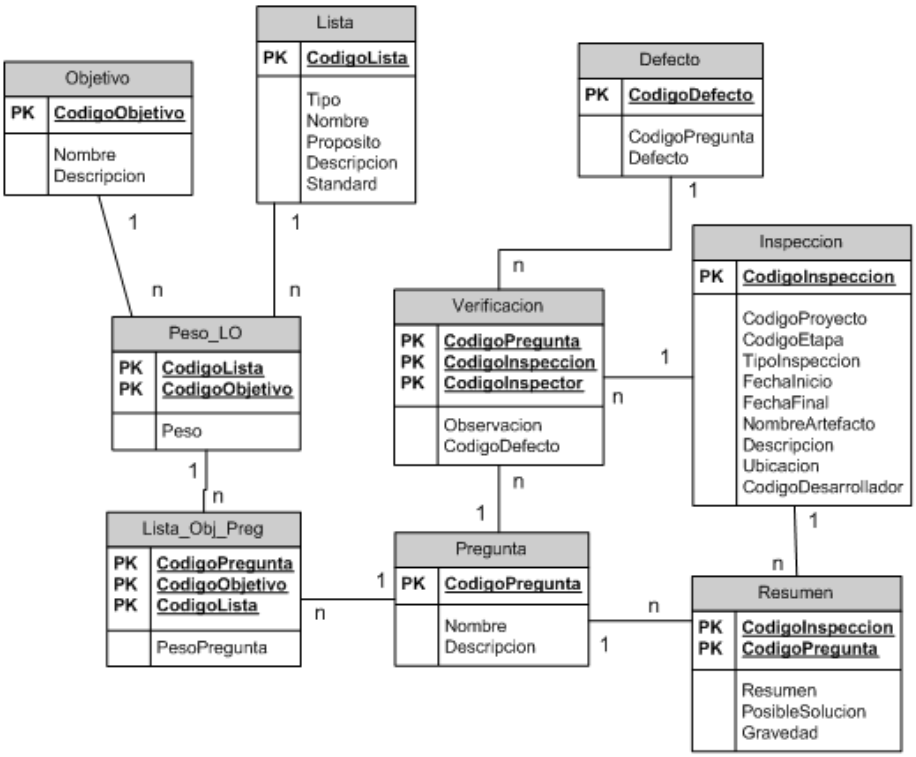


Figura 3.2. Modelo de Datos (Base de Datos)

3.9 ANÁLISIS DE COSTO/BENEFICIO DE LA APLICACIÓN

El análisis Costo/Beneficio determina los costos que se han incurrido en el desarrollo de la aplicación y los pondera con los beneficios tangibles e intangibles. El autor considera los siguientes beneficios que reducen el costo de desarrollo.

Beneficios Intangibles al aplicar la aplicación SPIS:

- Reducción de los costos por unidad de cálculo e impresión.
- Mayor precisión en las tareas de cálculo.
- Mayor velocidad de cálculo e impresión.
- Capacidad de almacenar “automáticamente” datos del proceso de inspección de software.
- Aumento de la seguridad de almacenamiento de los resultados de la inspección.
- Capacidad de hacer auditoria y analizar las actividades de búsqueda de registro.
- Capacidad de agregar grandes cantidades de datos útiles para la planificación y toma de decisiones.

- Reducción de la necesidad de fuerza de trabajo en el proceso de inspección de software.
- Capacidad mejorada de mantener una supervisión continúa de los recursos.

Haciendo uso de la herramienta COCOMO II de University of Southern California para el cálculo del costo de de la aplicación, se ha llegó a los siguientes resultados:

Esfuerzo	Cronograma	Productividad	Costo (Pesos Cubanos)	Personal
25.4	8.47	470.5	5,078.46	3

Tabla 3.1. Costo de la Aplicación

Para un total de líneas de código: 11,948.0

A pesar de que el costo ha sido considerable, el autor considera que los beneficios que se obtendrán, aunque no se han cuantificado, sufragaran los gastos. Además desde el punto de vista social ya se aprecia el valor de la aplicación en el proceso de Inspección de Software.

Para ver mayor detalle del cálculo de costo ver el Anexo E.

3.10 CONCLUSIONES

En el presente capitulo se ha descrito de forma general las facilidades y la arquitectura de la aplicación SPIS, dando a conocer, los requisitos funcionales y no funcionales, el patrón cliente/servidor, que esta compuesta por cuatro capas y la arquitectura de diseño (Anexo D).

Para finalizar el capitulo se ha realizado un análisis costo de la aplicación, dando como resultado 5,078.46 $ en moneda nacional.

CONCLUSIONES Y RECOMENDACIONES

Como conclusiones podemos señalar que los objetivos propuestos en el trabajo fueron alcanzados en su totalidad. Concretamente se puede afirmar:

- Se ha logrado establecer un modelo de inspección de software de acuerdo a las condiciones actuales, como consecuencia se ha obtenido un conocimiento sobre el proceso de inspección de software, logrando conocer el campo de acción y el porque debe aplicarse al desarrollo de software. El autor considera, que las inspecciones de software son una parte importante para encaminar a una empresa hacia el CMM (Modelo de Madurez de Capacidad).
- Se ha realizado una clasificación de las listas de comprobación de acuerdo a las fases de los proyectos tomando en cuenta los estándares internacionales. La clasificación, sirve para llevar a cabo, con éxito, el proceso de inspección de software, para de esta manera incrementar el conocimiento de los involucrados en el desarrollo de este, detectando los defectos y errores cometidos durante la elaboración. Además, con la utilización de las listas, se logra un conocimiento general de los artefactos que deben elaborarse para cumplir con los requisitos de calidad.
- Se ha conseguido, gracias al nuevo modelo y la clasificación, pasos específicos para utilizar las listas de comprobación, esto se refleja en la aplicación desarrollada que brinda facilidades para el manejo de la información generada por el modelo de inspección de de software. Además, se obtuvo una alternativa para la valoración de una lista de comprobación de acuerdo a los pesos de importancia que establece el Coordinador en el momento de planificar la inspección. Con esta alternativa, se observa si el proceso de construcción del artefacto se ha llevado correctamente, si lo planificado se ha logrado y valorar la capacidad de los involucrados en el desarrollo del artefacto.
- Se ha desarrollado una aplicación automatizada que refleja el modelo propuesto por el autor, logrando facilidades en la creación de listas de comprobación, el equipo de inspección, la agenda para llevar a cabo la inspección, la captura de los

defectos y dar apoyo a los inspectores en las distintas etapas del modelo para la captura de los defectos.

El autor considera que el trabajo propuesto permitirá al Grupo de Aseguramiento de Calidad llevar a cabo con mayor calidad el proceso de software, a través de la aplicación desarrollada con la ayuda de las listas de comprobación.

A continuación se presenta las recomendaciones, que a juicio del autor, es una propuesta sobre la continuidad del trabajo sobre el proceso de inspección de software:

- Implementar las distintas métricas de inspección de software, descritas en el Capitulo 2, dentro de la aplicación SPIS (Soporte al Proceso de Inspección de Software).
- Realizar un estudio estadístico que permita definir un modelo para los cálculos de los pesos, ya sea para los objetivos o preguntas de acuerdo a los resultados capturados por la aplicación SPIS. La definición del modelo debe ser incluida en la aplicación SPIS para la ayuda a la planificación de la inspección.
- Realizar una plantilla que debe estar reflejada en la aplicación, para la dirección de la empresa, que tome en consideración la información estadística sobre los defectos encontrados con vistas a facilitar la toma de decisiones para un mayor nivel de calidad de proceso de software, lo cual proporcionará una apropiada gestión del conocimiento.
- Elaborar una versión de listas de comprobación de acuerdo al proceso que siguen las distintas empresas cubanas.
- Realizar un modulo para la generación automática de los posibles defectos a encontrar en una etapa específica del proyecto o para una lista especifica.
- Realizar un modulo para el seguimiento a la corrección de los defectos encontrados y realizar un análisis del software existente sobre la Gestión de Configuración que pueda interactuar con este modulo.
- Se recomienda que la aplicación SPIS se utilice para la capacitación de líderes que lleven a cabo el proceso de inspección, tomando en cuenta las capacidades de dirección, administración y el carisma hacia los involucrados en el desarrollo.

REFERENCIAS BIBLIOGRAFICAS

[1] Jonás A. Montilva,
Universidad de los Andes
Departamento de Computacion,
Faculdad de Ingeniería,
http://www.ula.ve/DSIA/presentaciones/Mejoramiento%20de%20procesos%20de%20software%20CMM.ppt,
Venezuela.
Año: 2001

[2] Luis A. Guerrero
Universidad de Chile,
Departamento de Ciencia de la Computación
http://www.dcc.uchile.cl/~luguerre/cc51h/clase23.html
Chile.
Año: 2000

[3] Francisco Morfín Otero
Universidad Jesuita de Guadalajara
http://www.desi.iteso.mx/documental/mapa/m1.html
Mexico.
Año: 1998.

[4] Vladimir Estivill-Castro
Laboratorio Nacional de Informática Avanzada,
http://www.lania.mx/spanish/actividades/newsletters/1994-otono/art2.html
México.
Año: 1994.

[5] Gabriel Buades
Universidad de las Illes Balears,
Departamento de Ciencias Matemáticas e Informática,
http://dmi.uib.es/~bbuades/calidad/sld010.htm
España.

Año: 2002.

[6] Juan Manuel Cueva Lovelle
Universidad de Oviedo,

Departamento de Informática,

http://di002.edv.uniovi.es/~cueva/asignaturas/doctorado/2001/Calidad.pdf

España.

Año: 2001.

[7] Lawrence E. Hyatt
NASA,

Software Assurence technology Center,

http://satc.gsfc.nasa.gov/support/STC_APR96/qualtiy/stc_qual.PDF

U.S.A.

Año: 1996.

[8] Fundación Sidar,

Seminario de Iniciativas sobre Discapacidad y Accesibilidad a la Red

Métodos de Inspección,

http://www.sidar.org/visitable/index.htm,

España.

Año: 2002.

[9] European Laboratory for Particle Physics,

ATLAS DAQ/Event Filter Prototype -1 Project,

http://atddoc.cern.ch/Atlas/DaqSoft/sde/inspect/Introd_sw_inspection.html,

Suiza.

Año: 1999

[10] José Javier Dolado Cosín

Universidad del País Vasco,

www.sc.ehu.es/jiwdocoj/mmis/externas.htm,

España.

Año: 2001.

[11] Information System Audit and Control Association,

http://www.isaca.org.za/Iso9126.htm,

Africa.

Año: 2001.

[12] Garcia Romero,
Universidad de las América,
http://mailweb.udlap.mx/~tesis/lis/garcia_r_ci/capitulo1.pdf
México.
Año: 2001.

[13] Universidad de Cadiz,
Departamento de Lenguaje y Sistemas Informáticos,
http://www.monografias.com/trabajos5/call/call.shtml,
España.
Año: 1999.

[14] Tom Gilb
The Internacional Network of Software Process Improvement Consultants,
SPI Partners,
http://www.spipartners.nl/data/train_course_gilbinspect_en.html
The Netherlands.
Año: 2003.

[15] Linda L. Westfall
Software Division of the American Society for Quality,
Software Quality Web,
http://www.asq-software.org/metrics/aqc95.html.
Año: 1995

[16] Sergio Zapata
Universidad Nacional de San Juan,
Instituto de Informática,
http://www.idei-unsj.com.ar/Cursos/herra_mejora_desarr.htm,
Argentina.
Año: 2001

[17] Roger S. Pressman.
Ingeniería de Software Un Enfoque Práctico.
McGraw Hill.

Año: 2001

[18] Omar Lopez Ramos
Instituto Tecnológico de Cidudad Guzman
http://www.itcg.edu.mx/ingsoft/calidad.htm
Mexico.
año: 2003.

[19] Ilkka Tervonen,
University of Oulu,
Department of Information Processing Science,
http://www.cas.mcmaster.ca/wise/wise01/TervonenIisakkaHarjumaa.pdf,
Canada.
Año: 2001.

[20] Sun Sup So, Sung Deok Cha, Timothy J. Shimeall, Yong Rae Kwon
Software Engineering Laboratory at Kaist,
Division of Computer Science,
http://salmosa.kaist.ac.kr/LAB/sixmth-round5.doc,
Korea.
Año: 1999.

[21] Fraser Macdonald
The University of Strathclyde in Glasgow,
Department of Computer Science,
http://www.cs.strath.ac.uk/~efocs/home/abstracts.html#fm_thesis,
United Kingdom**.**
Año: 1998.

[22] The University of Strathclyde in Glasgow**,**
Computer and Information Sciences,
http://www.cis.strath.ac.uk/research/efocs/inspection.html**,**
United Kingdom**.**
Año: 2000

[23] W3C Resommendation
www.usabilidad.org/usablog/docs/Cap_V.doc

México.

[24] Juan Fuente
Universidad de Oviedo,
Departamento de Informatica. España,
www.di.uniovi.es/~aquilino/ficheros/articulos/Metricas/Sevilla-JJOO-1997.pdf
España.
Año: 1997.

[25] Building Web Aplications with UML
Jim Conallen
Addison – Wesley
Año: 2003

[26] Watts S. Humphrey.
The Personal Software Process
http://www.sei.cmu.edu/pub/documents/articles/pdf/psp.pdf
U.S.
Año: 1994.

[27] Department of Energy,
Chief Information Officer,
http://cio.doe.gov,
U.S.
Año: 2004

[28] Lawrence Livermore National Laboratory,
Departament of Energy,
University of California,
http://www.llnl.gov
U.S.
Año: 2004

[29] R. S. Pressman & Associates,
http://www.rspa.com
U.S.
Año: 2004

[30] Construx Software Builders,
http://www.construx.com
U.S.
Año: 2004

[31] Universidad de Stuttgart,
Facultad de Informática,
http://www.informatik.uni-stuttgart.de,
Alemania.
Año: 2003.

[32] Don O'Neill
Software Engineering Institute
http://www.sei.cmu.edu/str/descriptions/inspections_body.html
U.S.
Año: 2001

[33] Bazman
http://members.tripod.com/~bazman/checklist.html

[34] Daniel J. Mosley
Client-Server Software Testing (CSST)Technologies,
www.csst-technologies.com/cststar.htm
U.S.
Año: 1996.

[35] Charles Sturt University,
Division of Information Technology,
http://www.csu.edu.au,
Australia.
Año: 1999.

[36] Evoluciona an evol Group Company,
http://www.usabilidad.com,
España.

[37] Widener University,
Wolfgram Memorial Library,

http://www2.widener.edu/Wolfgram-Memorial-Library/webevaluation/ advoc.htm,
U.S.
Año: 1999.

[38] World Wide Web Consortium,
Technical Reports and Publications,
MIT Computer Science and Artificial Inteligence Laboratory, http://www.w3.org,
Cambridge, USA.

[39] Web Accessibility in Mind,
Utah State University,
http://www.webaim.org,
USA.
Año: 2001.

[40] University of Calgary,
The Departament of Coputer Science,
http://pages.cpsc.ucalgary.ca,
Canada.
Año: 1999.

[41] Usable Information Technology,
Jakob Nielsen's Website,
http://www.useit.com
Año: 1994

[42] ISO/IEC 9126 2001,
Information Technology - Software quality characteristics and metrics Part 1 - Quality characteristics and sub-characteristics,
Canada.

[43] Association for Computing Machinery,
http://www.acm.org,
USA.

[44] Dr. Dobb's,
Software Tolls for the Profesional Programmer,
United Busness Media,

http://www.webreview.com,
Año: 1999

[45] F. Macdonald, J. Miller, A. Brooks, M. Roper and M. Wood
The University of Strathclyde in Glasgow,
Computer and Information Sciences,
http://www.cis.strath.ac.uk/research/efocs/inspection.html,
United Kingdom.
Año: 1998.

[46] Jose Javiar Dorado y Luis Fernandez Sanz
Medición para la Gestión de la Ingeniería de Software,
Ra-Ma,
Año: 2000.

[47] University of Massachusetts Dartmouth,
Computer and Information Sceince Department,
www2.umassd.edu/CISW3/coursepages/pages/CIS311/LectureMat/sqa/sqa.html,
USA.
Año: 1997.

[48] Yuyu Wang,
Kansas State University,
Computing and Information Sciences,
http://www.cis.ksu.edu/SELab/SQA01/c3/841team.html,
USA.
Año: 2001.

[49] Rational Unified Process.
Rational Software Corporation. “Rational Unified Process”.
Versión 2003.06.00.65,
Copyright 1987-2003.

[50] Sistema de Centros Públicos
Centro de Investigación Matemática,
www.cimat.mx/~yolanda/cursos/semestre2/ingenieria/Proyecto/Presentaciones/PresentacionArquitecturas.pdf ,

México.
Año: 2003

[51] Don Mills
Document Quality Control by Inspection
www.spipartners.nl/data/insp_mills.zip
New Zealand
Año: 2000

[52] Elizabth Almeraz y Maria Pérez-Vargas
Avantare Consultores,
www.avantare.com/articulos/anteriores

[53] Gordon Schulmeryer and James McManus,
Handkook of Software Quality Assurance,
Prentice Hall
Año: 1999

[54] Software Engineering Institute, Carnegie Mellon University
Capability Maturity Model® Integration (CMMI℠) Version 1.1
http://www.hi.is/pub/cs/2001-02ms/gsh/cmmi/v1-1/
Año: 2002

[55] Juan Manuel Cueva Lovelle
Universidad de Oviedo,
Departamento de Informatica. España,
http://www.di.uniovi.es/~cueva/asignaturas/doctorado/2001/Calidad.pdf
España
Año: 2001

[56] Paulk, Weber, Curts, Beth Chrissis
Carnegie Mellon University
Software Engineering Institute
The Capability Maturity Model: Guidelines for Improving the Software Process
Addison-Wesley
Año: 1999

[57] Claudia Ivette García Romero

Universidad de las Américas-Puebla
El Modelo de Capacidad de Madurez y su Aplicación en Empresas Mexicana de Software
http://mailweb.udlap.mx/~tesis/lis/garcia_r_ci/
Año: 2001

[58] Universidad Jesuita de Guadalajara
Departamento de Electrónica, Sistemas e Informática
Francisco Morfín Otero
http://www.desi.iteso.mx/documental/mapa/m1.html
Año: 1999

[59] ISO-15504,
Valoración del Proceso de Software Parte 1.
Canada.
Año: 1999

[60] Iván Cifuentes
Universidad de los Andes
Estado del Proceso de Desarrollo de Software en Colombia
http://agamenon.uniandes.edu.co/sistemas/6707.htm
Colombia
Año: 1996

[61] Centro de Comercio Internacional
Una Introducción a la ISO 9000: 2000
http://www.prompex.gob.pe/prompex/Certificadoras/iso9000.pdf
Perú.
Año: 2001

[62] Carnegie Mellon University
Software Engineering Institute
Capability Maturity Model Integration
Staged Representation
http://www.sei.cmu.edu/cmmi/models/v1.1ippd-staged.doc

Año: 2001

[63] Carnegie Mellon University
Software Engineering Institute
Capability Maturity Model Integration
Continuous Representation
http://www.sei.cmu.edu/cmmi/models/v1.1ippd-cont.doc
Año: 2001

[64] Department of Defence Information Analysis Center,
Using Cost Benefit Analyses to Develop Software Process Improvement (SPI) Strategies
http://www.dacs.dtic.mil/techs/RICO/toc.shtml ,
USA.

[65] Microsoft Visual Studio .NET,
MSDN Online en Español,
www.microsoft.com/spanish/msdn/vstudio/productinfo/vstudio03/features/edfeatures.asp .
Año: 2003.

[66] Rational the e-development Company,
The UML and Data Modelling A Rational Software Whitepaper,
Rational,
Año: 2000

[67] Thomas Connolly, Carolyn Begg, Anne Strachan,
Database Systems - A Practical Approach to Design, Implementation, and Management,
Addison-Wesley,
Año: 2002

[68] Fernández Humberto,
Procesos del Software
[tesis de maestría],
Ciudad de La Habana: Instituto Superior Politécnico "José Antonio Echevarria",

Año: 2003.

BIBLIOGRAFIA

[69] Derrick Story,
Usability Checklist for Site Developers,
http://www.webreview.com/archives/usability.htm
Año: 1999

[70] Gloria Nistal Rosique
Calidad del Software
http://www.ati.es/novatica/1999/137/pres137.html
España,
Año: 1998.

[71] Jakob Nielsen,
Using Discount Usability Engineering,
http://www.useit.com/papers/guerrillaHCI.htm
U.S.
Año: 1994.

[72] The Internacional Network of Software Process Improvement Consultants,
SPI Partners,
www.spipartners.nl/data/insp_mills.zip,
Netherlands

[73] John Frankovich
The Software Inspection Process
http://sern.ucalgary.ca/courses/seng/621/W97/johnf/inspections.htm
Canada.
Año: 1994

[74] M. Halling, P. Grünbacher, S. Biffl,
Groupware Support for Software Requirements Inspection,
http://www.cas.mcmaster.ca/wise/wise01/HallingGrunbacherBiffl.pdf,
Canada,
Año: 2001

[75] Jim Conallen,
Building Web, Applications with UML,
Addison-Wesley,
Año: 2003