UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE INGENIERÍA

TESINA

DISEÑO Y DESARROLLO DE LA INFRAESTRUCTURA DE PRUEBAS PARA EL SISTEMA TIME REPORT

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

EMANUEL HERRERA CERDA

ASESOR:

M.I. TANYA ITZEL ARTEAGA RICCI

Enero 2016

CIUDAD UNIVERSITARIA, D. F.

ÍNDICE

Índice de figuras

INTRODUCCIÓN

La recopilación de este documento es con el fin de dar a conocer las tareas que he realizado en mi desempeño profesional, para cumplir con las diferentes actividades orientadas al *testing*, para la empresa donde laboré.

Con el paso del tiempo las ciencias de la informática y los sistemas computacionales han ido evolucionando y se han extendido abarcando nuevas áreas de desarrollo, como sería el área de *testing*, el cual no es un concepto nuevo, ya que a pesar que desde las primeras generaciones de computadoras se han aplicado métodos muy parecidos al *testing*, estas técnicas no eran reconocidas y no tenían estandarización, tampoco se realizaban buenas prácticas para las pruebas, hoy en día debido a las nuevas demandas de producir software con pocos errores y una alta calidad en la etapa de distribución del sistema o software en el mercado para el consumidor, el *testing* se ha vuelto parte fundamental de las buenas prácticas para las empresas.

En este trabajo se abordarán las actividades que me fueron asignadas y desarrolle como *tester* para el proyecto Time report (TR).

Dicho proyecto fue creado para solventar la necesidad especifica del cliente con el objetivo de llevar una administración y mejor control de los recursos humanos.
Cada colaborador de la empresa del cliente podrá capturar de manera personal las horas laboradas por medio de una interfaz sencilla y fácil de operar, con lo anterior se podrá llevar una gestión altamente confiable de la participación de dichos recursos en los diferentes proyectos que el cliente está desarrollando.
Se hablará más detalladamente del TR en el capitulo 3.
En el resto de los capítulos se hará un análisis sobre la documentación necesaria y las actividades más relevantes del caso específico de *tester* además se abordará la importancia de llevar a cabo un orden para lograr alcanzar los objetivos planteados.

OBJETIVO GENERAL

Satisfacer los estándares de calidad del software desde la fase inicial de análisis, diseño y desarrollo hasta su respectiva entrega y soporte del sistema al cliente.

OBJETIVO ESPECÍFICO

Por medio del análisis para el diseño y desarrollo de casos de pruebas se establece una metodología robusta orientada a la prevención, detección, verificación y validación de errores, defectos o fallas que pueden afectar de forma negativa la calidad y desempeño del sistema a desarrollarse.

Se proporciona una infraestructura con la capacidad de facilitar la detección y generar reportes de las diferentes anomalías que puedan manifestar un impacto negativo al sistema en desarrollo y a la empresa.
De esta forma se podrán prevenir y solucionar para entregar un producto funcional y fiable.

CAPÍTULO 1

DESCRIPCIÓN DE LA EMPRESA

1.1 ACERCA DE LA EMPRESA

MTP & TOWA Consulting fue fundada a partir de la asociación de TOWA y MTP en el año 2004.

MTP nace como empresa mexicana en 1998. Fue creada como una empresa proveedora de servicios de consultoría y software, dirigida a apoyar a las medianas y pequeñas empresas en sus necesidades de desarrollo tecnológico.

TOWA nace en el año 2004, uno de los fundadores de Softtek creó una productora de software basada en la calidad dándole el nombre de TOWA, con el pensamiento de utilizar la ingeniería para ejecutar proyectos de software con certeza, TOWA se ha dedicado a la investigación y desarrollo de procesos ingenieriles de análisis y construcción de sistemas de información. Logrando mediante sus técnicas TSP (team software procces) y PSP (personal software procces) redefinir la estructura para desarrollar software por medio de procesos institucionales enfocados a la ejecución de las tareas ingenieriles, orientándose en adaptarse a las diferentes tecnologías que sus clientes requieran.

Tras 6 años de operación MTP se asocia con la empresa regiomontana TOWA software con el fin de fortalecer la experiencia en los servicios de consultoría e implementación de soluciones empresariales mejorando la atención de sus clientes en la zona norte del país.

MTP & TOWA Consulting socio de negocios de Microsoft Business Solutions se distingue por ser una compañía innovadora en metodologías y procesos para la implementación de sistemas.

i) Fig. 1.1: Logo de la empresa

1.2 PROYECTOS, ÁREAS DE DESARROLLO Y ESQUEMA LABORAL DE LA EMPRESA

MTP & TOWA al ser una empresa orientada al desarrollo de las nuevas tecnologías, debe contar con un amplio esquema laboral para los diferentes proyectos y sus áreas de desarrollo, las cuales se pueden definir como:

- Proyectos de tiempo y material.
- Proyecto de costo fijo (Instalaciones del Cliente o TOWA).
- Soporte y Mantenimiento de Aplicaciones (*AMS*).
- Mantenimiento de aplicaciones existentes y residentes en producción.
- Flexibilidad para aumentar o disminuir el equipo de trabajo a demanda. Basado en la conformación de un *core-team* que conoce profundamente el negocio, orientado al cumplimento de niveles de servicio establecidos contractualmente.

Proyectos de Software con certeza

Resaltando los siguientes puntos para un desarrollo de software a la medida:

- Sin Defectos
- Predecibles
- Costo Total de Propiedad (TCO) más bajo

Análisis (Proceso de Definición de Requerimientos)

Resaltando los siguientes procesos para el análisis:

- Eliminación de la brecha entre la tecnología y el negocio
- Requerimientos basados en actividades de negocio
- Definición de requerimientos a distancia

Proyectos

Resaltando los siguientes puntos:

- Desarrollo software nuevo
- Adaptación de software existente
- Perfeccionamiento software para alto mantenimiento

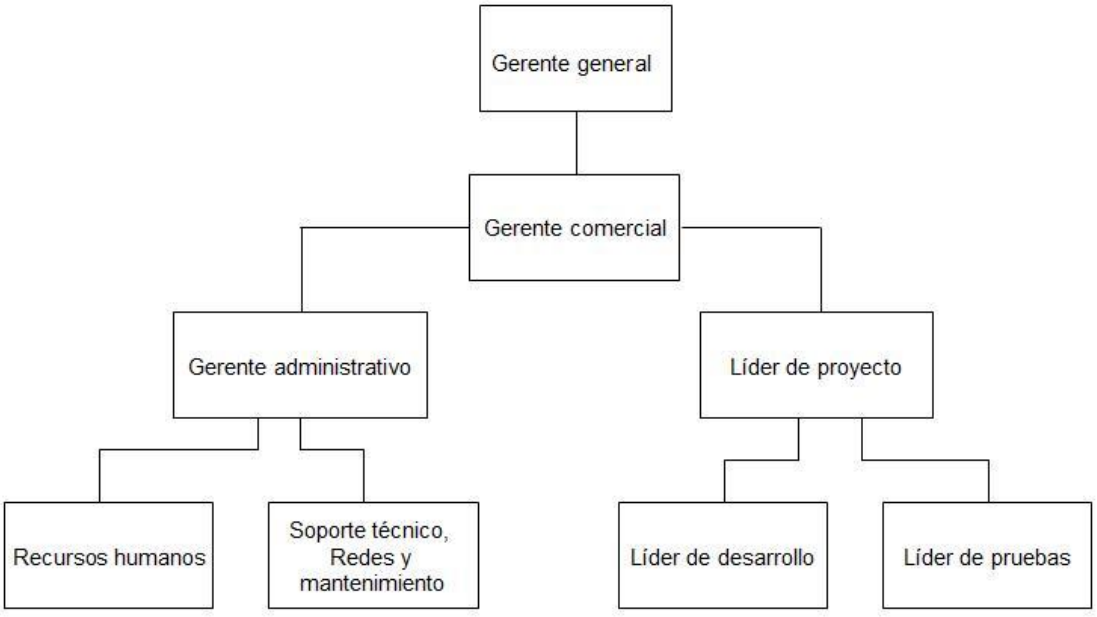
Proteger a nuestros clientes

Resaltados los siguientes puntos para la protección del cliente:

- Proteger la propiedad intelectual
- Reducir la dependencia del ingeniero de software
- Flexibilidad en la asignación de recursos a proyectos del cliente

1.3 ORGANIGRAMA DE LA EMPRESA

Estructura jerárquica de las diferentes áreas que forman el organigrama de la empresa, el área de pruebas es donde desempeñe mis diversas actividades como tester.



ii) Fig. 1.2: Organigrama de la empresa

CAPÍTULO 2

E200 – BUSINESS CASE

El alcance del proyecto es parte del objetivo y de la suma de todos los requerimientos postulados por el cliente, donde se involucran los detalles y las características del sistema a desarrollar.

Este documento identificado con la nomenclatura de TOWA (E200) es creado a petición del cliente y nos sirve para definir las implementaciones que se llevarán a cabo en el proyecto por medio del equipo de desarrolladores y los *testers*.

El cliente es el que denota el alcance, y todo lo que se debe abarcar desde el inicio hasta el final de éste.

2.1 SITUACIÓN ACTUAL: PROBLEMAS Y NECESIDADES DETECTADOS

En este módulo se enuncian las necesidades del cliente, donde se define la situación actual de los problemas de su empresa y las necesidades que este requiere para solucionar la situación de la empresa.

El E200 describe las especificaciones para el desarrollo del proyecto a implementar, el cual debe ser un sistema orientado a la administración del personal que labora en la empresa del cliente, esto radica en la necesidad del cliente en implementar un sistema que unifique el control y administración de horas y recursos que laboran en la empresa.

El proyecto debe ser un sistema con la capacidad de soportar una gran cantidad de usuarios que laboran en la empresa y que estos puedan ingresar a cualquier hora por medio de una interfaz a través de una red interna para que en estos puedan capturar sus horas laboradas en los diferentes proyectos en los que participaron y trabajaron dentro de la empresa del cliente.

En la gestión de proyectos tradicional, las herramientas para describir el alcance del producto de un proyecto es la descripción del mismo. El proyecto al ser un sistema que afecta a todo el personal que labora en la organización del cliente, y que además involucra procedimientos monetarios para el pago de algún servicio, es de suma importancia garantizar que el sistema tenga un funcionamiento eficiente y correcto.

Un factor importante antes de que el sistema entre a producción, es que se cumpla todo lo postulado en el alcance del proyecto, ya que se asocia a las reglas de negocio y los requerimientos del cliente, el alcance del proyecto como tal no es algo que se define de forma resumida, ya que parte del objetivo general y los requerimientos solicitados por el cliente involucrando costo, tiempo y funcionalidad.

2.2 INFORMACIÓN DE LA INICIATIVA Y REQUISITOS

El sistema debe cubrir las siguientes iniciativas y requisitos:

1. Cargar Transacciones: Módulo de donde se debe cargar documentos en formato Excel que serán generados en SPG para las siguientes transacciones:

- PEP
- Órdenes de compra
- Aceptaciones de PEP

2. Asignar Responsable: El jefe inmediato de un proyecto de la empresa tendrá asignado un PEP, este jefe inmediato podrá asignar permisos a un recurso activo externo o interno que tendrá la función como "Responsable" a partir de un nivel inferior de la estructura del Jefe inmediato, para que éste pueda apoyar al Jefe inmediato en la asignación de sus recursos activos a sus respectivas órdenes de compra.

3. Asignar recurso externo a Orden(es) de Compra: El jefe inmediato o el responsable podrá asignar cada recurso externo, que estén en el nivel inferior de la estructura donde se encuentra el jefe inmediato por el total de 172 horas por periodo. Se puede hacer la búsqueda por:

- Recurso externo
- Orden de compra

4. Proceso de administración de bitácoras: Este módulo contempla:

- Proceso de Generación Automático de Bitácoras
- Proceso de Autorización/Rechazo de Bitácoras
- Proceso de "Congelación" Automático de Bitácoras
- Proceso de Autorización de Bitácoras rechazadas
- Proceso de Autorización automático de bitácoras rechazadas
- Consulta de Bitácoras
- Impresión de la bitácora
- Generar reporte en excel de bitácoras

5. Generación de Reportes: El sistema debe contar con un módulo para generar los siguientes reportes:

- Reporte de Asignaciones. En este reporte se podrá consultar en qué orden de compra se encuentra asignado qué recurso externo.

- Reporte de Aceptaciones. En este reporte se podrá consultar por *PEP*/Orden de compra/Posición, principalmente, las horas aceptadas en SPG y las horas autorizadas en Bitácora.

6. Administración de recursos externos TR-SIAPRO (Alta, baja y cambio)

SIAPRO es un sistema interno de la empresa del cliente con la funcionalidad de administrar el estado o la situación actual de los recursos humanos externos que prestan sus servicios o laboran para la empresa del cliente, este sistema indica si el recurso externo esta dado de alta en la empresa del cliente, si este está dado de baja o si este se cambio a otra área dentro de la empresa del cliente.
Esta meta tiene como propósito evitar la doble captura de los recursos externos, manteniendo la información actualizada y consistente entre TR y *SIAPRO* a través de un proceso automático que hará uso de vistas entre ambos sistemas.

A petición del cliente se necesita que los siguientes *requerimientos* sean atendidos:

- Time report proporcionará la vista de "Jefes inmediatos" (nombre y usuario de red) a *SIAPRO* para que se vinculen en el registro del recurso externo al momento del alta de recurso externo en *SIAPRO*.

- Las altas, cambios (de datos personales del recurso externo, de Perfil y de "Jefe inmediato") y bajas definitivas serán ejecutados en *SIAPRO*.

- A través de los reportes de asignación se podrá conocer en que órdenes de compra se encuentran asignados los recursos externos.

- Con el reporte de bitácoras se podrá consultar las horas facturables, las horas no facturables, el total de horas por pagar y el total de horas autorizadas en bitácora del recurso externo.

- Con el reporte de aceptaciones se conocerá a nivel PEP/órdenes de compra/posición las horas asignadas, las horas sin asignar, las horas autorizadas, las horas sin autorizar, las horas aceptadas en SPG y las horas sin aceptar en SPG, para tener un mejor detalle de lo que se trabajó vs lo que se facturó.

2.3 RESTRICCIONES

Se mencionan las restricciones al momento de la puesta en operación del sistema:

La asignación de recursos externos a órdenes de compra se realizará a partir de la fecha puesta en operación del sistema y no se podrán asignar recursos externos a pasado porque no se dispone de la información de las estructuras previas.

Las bitácoras se generarán a partir del primer cierre de operación de un periodo, considerando la fecha de puesta en operación del sistema.

Algunas órdenes de compra a las que se estén asignados recursos externos, quedarán con horas disponibles que correspondan a recursos externos que trabajaron antes de la fecha de puesta en operación del sistema.

Hasta un año posterior al mes de la puesta en operación del sistema, se dispondrá de la información completa de los 12 meses anteriores.

A partir de que se tengan los 12 meses anteriores, el sistema realizará una depuración automática de los registros anteriores a estos 12 meses.

Si se pagaron recursos externos con órdenes de compra del periodo/año anterior, estos recursos no se podrán asignar ya que no se contará con las órdenes de compra al momento de la puesta en operación del sistema.

Los reportes (aceptaciones, asignaciones y bitácoras) contarán con información relativa a la fecha de la puesta en operación.

Debe existir una persona responsable de generar las transacciones de PEP, Órdenes de Compra y Aceptaciones en SPG.

CAPÍTULO 3

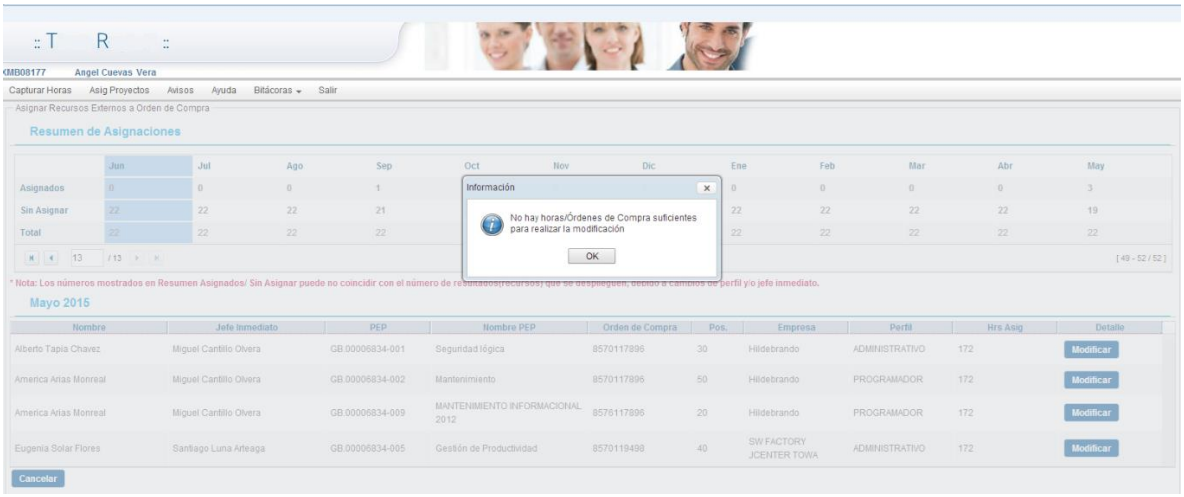
DESARROLLO DEL SISTEMA TIME REPORT (TR)

3.1 TIME REPORT

Time report (TR) es el nombre del proyecto que se desarrolló, es un robusto sistema implementado a partir de la plataforma ZK(framework de aplicaciones web en java) el cual brinda una herramienta de soporte para el diferente tipo de personal que labora y ofrece algún servicio a la empresa del cliente, para que puedan capturar sus horas laboradas en alguno de los diferentes proyectos en los que estuvieron involucrados por medio de una red privada interna *LAN*, accediendo a ésta mediante el uso de un ID de usuario y una contraseña, por este medio se le puede asociar una forma de pago por sus respectivos proyectos y servicios. La complejidad del sistema radica en los diferentes roles que pueden tener los usuarios y los diferentes tipos de proyectos a los que se les asocia un recurso. El sistema considera todas las jerarquías a nivel usuario y sus respectivos roles como jefes inmediatos, administrador Time report, recursos internos y externos, los diferentes tipos de proyectos como proyectos cerrado, PSP-TSP (PSP personal software process) y TSP (team software process), *Adaevo* y propio.

El sistema abarca una gran funcionalidad a nivel de usuario dependiendo de su respectivo rol, es decir los usuarios con menor jerarquía como los empleados solo utilizarían el sistema para poder capturar sus horas laboradas por día por lo cual solo personas con mayor jerarquía como un administrador podrá visualizar todo el panel de las diferentes acciones y funcionalidades que puede efectuar el sistema.

Esta es la interfaz visual de time report



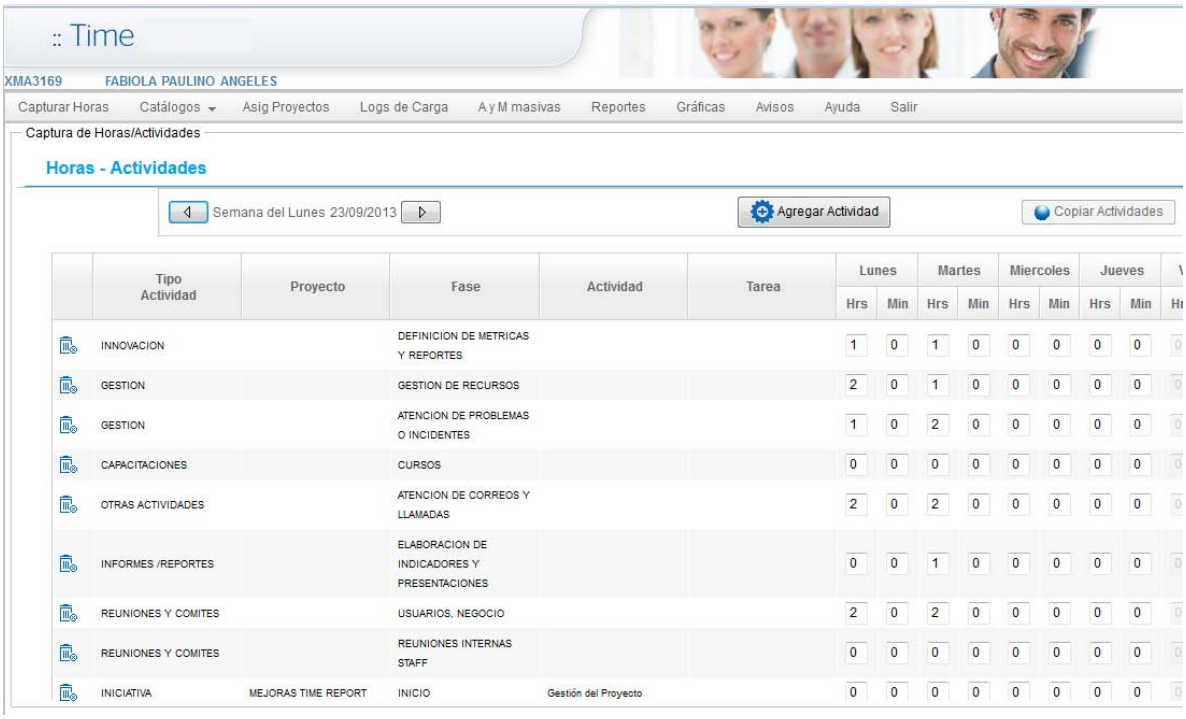
iii) Fig. 3.1: Imagen del sistema Time report

3.1.1 MEJORAS Y MODIFICACIONES EN TIME REPORT

Time report (TR) era un sistema funcional incompleto, ya que no todos sus módulos tenían alto rendimiento en sus diferentes funcionalidades, necesitaba adaptarse a nuevos proyectos, nuevas mejoras y nuevos tipos de recursos. Las modificaciones que requiera el cliente para Time report se mencionan en los diferentes puntos:

Unificar el control de la administración de horas y recursos, entre los 3 sistemas ya existentes;

Time. - captura de horas de recursos internos y externos.

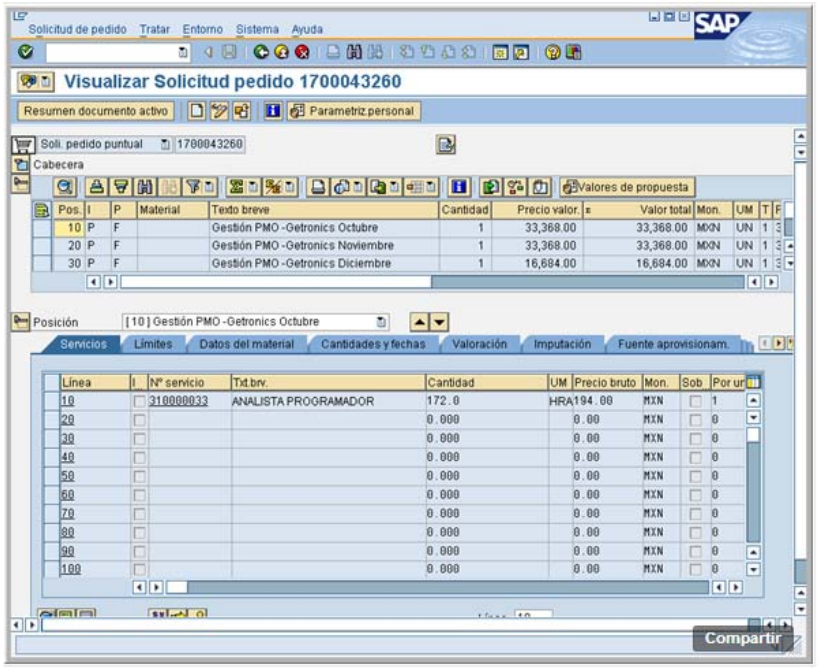


iv) Fig. 3.2: Visualización de la interfaz de Time

Bitácoras. - captura de horas que serán facturadas al proveedor.

v) Fig. 3.3: Visualización de la interfaz de Bitácoras

SPG. - Sistema para la gestión y control de *PEP*, solicitudes y órdenes de compra.



vi) Fig. 3.4: Visualización de la interfaz del SPG

Poder cargar información de las horas incurridas por recursos externos e internos en proyectos PSP/TSP tipo “P”, “I” y recursos Tipo Empresa para proyectos PSP/TSP “C” y Proyectos Cerrados para evitar la doble captura de horas, así como la asociación de horas incurridas en mantenimiento al proyecto Genérico Mantenimiento.

En algunos casos la metodología para capturar horas cambiaba dependiendo del tipo de proyecto y del recurso, los recursos que pertenecen a un proyecto cerrado no podían capturar en Time report de forma directa las horas incurridas en el proyecto, a diferencia de un recurso asignado a un proyecto *Adaevo*, en caso de que el recurso pertenezca a un proyecto cerrado se implementó un módulo donde su respectivo jefe inmediato adjuntaba un archivo Excel donde el sistema validaba los datos por celda y almacenaba los resultados en una base de datos centralizada donde se asociaba al usuario con su respectivo proyecto y las respectivas horas asociadas a los proyectos cerrados. También se debía considerar que los diferentes tipos de usuarios puedan no pertenecer a todos los proyectos, es decir un recurso externo puede pertenecer o asociarse a un *proyecto cerrado*, pero este no puede pertenecer a un proyecto PSP/TSP tipo I. De igual forma el sistema tenía que pasar por algunas validaciones antes de concretar el *PEP* (determinada cantidad de dinero asignada a un proyecto). Una vez asociado el *PEP* con el proyecto se genera una bitácora, la cual el jefe inmediato autoriza o rechaza, una vez autorizada la bitácora el administrador puede visualizar las bitácoras autorizadas y no autorizadas que se generaron asociadas a su respectivo mes. Y en caso de que se quiera cancelar una bitácora autorizada por un jefe inmediato solo el administrador Time report puede desautorizarla.

El desarrollo presentó todo un reto para el equipo de pruebas, esto pasó debido a que se tenía que trabajar con un sistema que fue elaborado anteriormente por otro fabricante de software, el proyecto quedo funcional sin embargo al entrar a producción con el cliente, este software resultó no ser eficiente en algunos módulos, debido a que el anterior desarrollador carecía de toda una metodología de infraestructura para el desarrollo y la implementación de pruebas, por lo cual, el proyecto se retomó solo que ahora para realizarle mejoras y además de unificar el control de la administración de horas y recursos, entre los 3 sistemas ya existentes(Time, SPG y Bitácoras) .

Debido a esta circunstancia existía cierta incertidumbre entre la dependencia de algunos módulos ya desarrollados y las nuevas modificaciones que se tenían que implementar ya que podían o no afectar algún módulo, ya sea de forma interna o de forma dependiente de otro módulo, afectando parte de la funcionalidad del código y del sistema, el reto para el equipo de desarrollo fue trabajar en las mejoras de módulos dependientes, sin afectarlos de forma negativa, sobre todo a aquellos donde no se requería algún cambio o algún tipo de intervención.

3.2 EQUIPO DE PRUEBAS

Para mejorar la funcionalidad del *software* la empresa creó el área de pruebas, con el fin de implementar estrategias de *testing* para revisar y asegurar la integridad del producto.

El equipo de pruebas está compuesto por un grupo selecto de personas orientadas a un mismo fin, garantizar la calidad del producto por medio de sus roles profesionales, realizando labores competentes para el desarrollo del área de *testing*.

3.2.1 EQUIPO DE PRUEBAS ORIENTADO A JAVA ZK PARA TIME REPORT

ZK es un *framework* de aplicaciones web en AJAX, el cual nos permite definir la gramática de lenguajes específicos (de la misma manera que HTML), es el marco abierto de Java para construir páginas web con software de código abierto, el cual contiene una completa interfaz de usuarios para aplicaciones web, ZK utiliza el enfoque centrado en el servidor para la sincronización de componentes y el *pipelining* entre clientes y servidores de forma automática, además de que los códigos de AJAX sean completamente transparentes para los desarrolladores de aplicaciones web. Por lo tanto, los usuarios finales obtienen una interacción y respuesta similar a las de una aplicación de escritorio, mientras que la complejidad del desarrollo es similar a la que tendría la codificación de aplicaciones de escritorio.

Una desventaja y a la vez ventaja es que ZK no es muy conocido para el desarrollo web, lo cual brinda un cierto nivel de seguridad, pero también como desventaja está el hecho que no es fácil encontrar información para el desarrollo y soporte de programas realizados en ZK.

El equipo de pruebas contó con el apoyo del equipo de desarrollo el cual nos brinda una ligera introducción a algunas funcionalidades de ZK, a pesar de que el equipo de pruebas no programó en ZK, teníamos una noción del lenguaje de programación lo cual nos ayudó a conocer de mejor forma el comportamiento del sistema para realizar las pruebas.

El reto del equipo de pruebas para este producto fue el trabajar con una interfaz tan poco conocida como lo es ZK.

3.3 PATRON VISTA – MODELO DE VISTA (MVVM) IMPLEMENTADO EN ZK

El equipo de desarrollo se basó en el modelo (MVVM) para el diseño y la estructura del proyecto TR desarrollado en ZK, este modelo automatiza las tareas obligatorias de datos y se basa en crear un modelo visual, un modelo de datos y un controlador el cual sirve como acción que une los modelos anteriores.

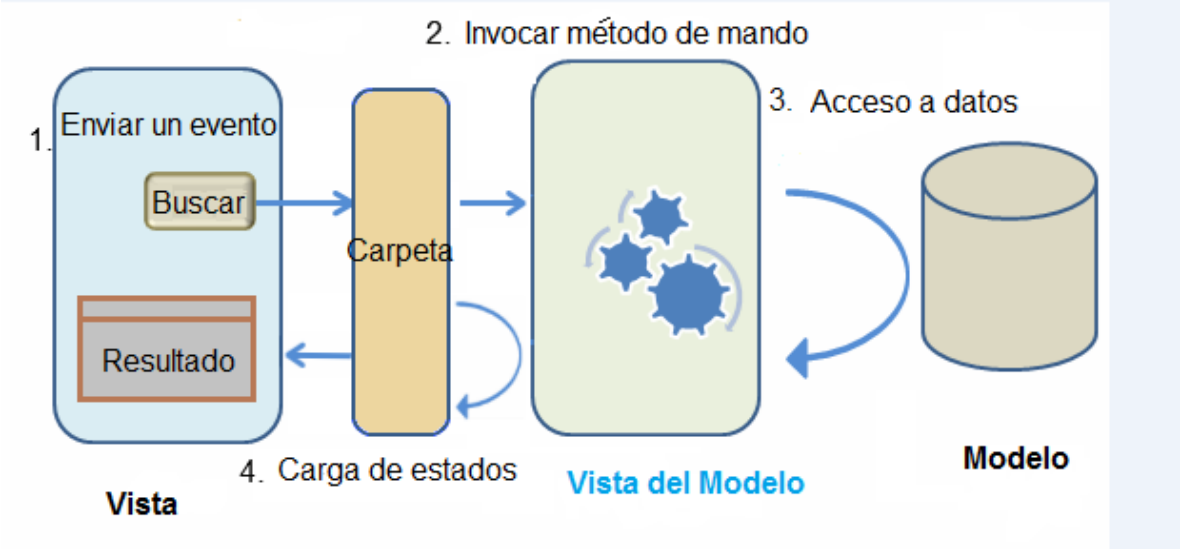
Este modelo divide un uso sobre tres partes.

El modelo. - Este representa los datos o la información con la que trabaja el equipo de desarrollo el cual se le refiere como un objeto de dominio, este modelo contiene la información, pero no las acciones o servicios que manipula. No es responsable de darle forma a los objetos que se muestran en la vista o interfaz.

La vista. - Esto representa la parte visual o gráfica. Con la que los usuarios están más familiarizados. Esta vista consiste en representar la información además de contener ciertos comportamientos como las entradas de datos.

Modelo de Vista. - Este modelo hace disponibles métodos, comandos y otros puntos de acceso que mantienen el estado de la vista, para manipular el modelo en respuesta a acciones de este y disparar eventos.

El modelo vista mantiene al modelo separado para limitar y contener los datos.



vii) Fig. 3.5: Modelo-Vista-Modelo de Vista

3.4 HABILIDADES TÉCNICAS DE UN TESTER

Un *tester* debe conjugar un grupo de habilidades relevantes para su desempeño profesional en la empresa, como:

- Conocimiento del objetivo para los requerimientos del cliente.
- Conocimiento de la aplicación o sistema bajo el entorno de pruebas, de cómo planificar, diseñar, ejecutar y administrar las pruebas.
- Análisis y diseño para planificación de casos de prueba orientados al negocio y al desarrollo.
- Conocimiento para el modelado de diagramas de lenguaje unificado para entender el comportamiento del sistema bajo prueba.
- Capacidad para abstraer e identificar comportamientos diferentes al resultado esperado.
- Decisiones adecuadas y asertivas de las diferentes herramientas para resolver un determinado problema.
- Detectar y contemplar la mayor cantidad de fallas, antes de que el software salga a producción.
- Conocimiento conceptual de lenguajes de programación, base de datos y sistemas operativos.
- Conocimientos de diversas plataformas, IDE´s y sistemas operativos.
- Capacidad para comprender y aprender nuevas tecnologías.

3.4.1 HABILIDADES SOCIALES DE UN TESTER

Un *tester* debe tener ciertas habilidades sociales para poder desempeñarse de forma eficiente en su entorno laboral, considero que estas son algunas cualidades que un *tester* debe tener:

- Facilidad de comunicación oral y escrita para interactuar con desarrolladores y usuarios.
- Creatividad para generar ideas e imaginar los problemas que podrían existir o presentarse.

- Pensamiento crítico para evaluar las ideas, hacer deducciones y vincular lo observado con los criterios de calidad de la empresa.

- Pragmatismo para poner en práctica las ideas y adecuar las técnicas y el esfuerzo al alcance del proyecto.

- Evaluación de las tareas proporcionadas en tiempos de acuerdo a la entrega estimada en determinada fecha calendarizada.

- Aptitudes para el trabajo en equipo, de manera que se pueda interactuar con los desarrolladores y otros *testers*, para lograr el máximo beneficio en esta interacción.

- Paciencia para poder enfrentar situaciones que no habían sido realizadas con anterioridad

3.4.2 HABILIDADES COGNOSCITIVAS E INTELECTUALES

Un *tester* debe conjugar un grupo de habilidades orientadas a la capacidad de entender y comprender las labores que debe realizar haciendo énfasis en su función intelectual para una mejor comprensión del proyecto, algunas de estas habilidades se pueden describir como:

Observar. - para dar una dirección intencional a la percepción y habilidades como *concentrarse, identificar, buscar y encontrar datos, elementos u objetos.*

Analizar. - para destacar los elementos básicos de una unidad de información como *comparar, distinguir, resaltar.*

Ordenar. - de manera sistemática un conjunto de datos, a partir de un atributo determinado. Ello implica habilidades como *reunir, agrupar, listar.*

Clasificar. - para disponer o agrupar un conjunto de datos según categorías. Como por ejemplo, *jerarquizar, esquematizar, categorizar.*

Representar. - para la recreación de nuevos hechos o situaciones a partir de los ya existentes. Como son *simular, modelar.*

Memorizar. - implica procesos de codificación, almacenamiento y recuperación de una serie de datos. Este hecho supone también *retener, conservar, archivar, evocar, recordar.*

Interpretar. - para atribuir significado personal a los datos contenidos en la información recibida. Interpretar implica habilidades como *razonar, argumentar, deducir, explicar, anticipar.*

Evaluar**. -** consiste en valorar a partir de la comparación entre un producto, los objetivos y el proceso. Involucra habilidades como *examinar, criticar, estimar, juzgar.*

3.4.3 PENSAMIENTO CRÍTICO Y CREATIVIDAD

El perfil de un *tester* debe estar vinculado a desarrollar un pensamiento crítico para evaluar las ideas, hacer deducciones y asociar lo observado con los criterios de calidad de la empresa, pero de igual forma debe desarrollar la creatividad para generar ideas e imaginar los problemas que podrían existir o presentarse.

Un aspecto fundamental en el cual se puede medir la capacidad de creatividad de un individuo consiste en dar respuestas que son insólitas u originales, pero a la vez deben ser correctas y efectivas para solucionar un problema o una pregunta.

Un *tester* también debe tener capacidad de pensamiento convergente, ya que, si la respuesta para solucionar un problema es muy obvia o muy sencilla, no será necesario buscar una respuesta muy creativa e innovadora cuando se puede dar una solución rápida, corta y efectiva.

La creatividad juega un papel importante para el diseño y la estructura de los casos de prueba. Pero también se necesita tener un pensamiento crítico el cual debe estar basado en tener un buen entendimiento de los requisitos, la funcionalidad y las reglas de negocio. Ya que estas serían las bases para poder crear casos de prueba orientados a la funcionalidad del sistema.

CAPÍTULO 4

DESCRIPCIÓN DE LAS ACTIVIDADES

4.1 DEFINICIÓN Y ESPECIFICACIÓN DEL DISEÑO

Para la descripción de las actividades que se desarrollaron, se necesitaba realizar un documento a especificación del cliente para el diseño visual del sistema, aquí se define la estructura visual, donde se involucran las imágenes, logos, colores, tipo de letra y todo lo que se relaciona con el diseño.

Además, en este documento, se definen las funcionalidades de cada parte y componente visual de la estructura del sistema, es decir, si el cliente define un título con el nombre de su empresa y logo en la parte superior, se especifica en el documento que ese título hace referencia a la posición exacta, con el tipo de letra, el color de la letra, tamaño, ubicación, etc.

Cuando se ha terminado la tarea de recopilación de las definiciones y especificaciones que el cliente solicita, además de la aprobación por parte del cliente de este documento se puede pasar a la siguiente etapa de producción.

4.1.1 FLUJO DE CREACIÓN DE LA DEFINICIÓN Y ESPECIFICACIÓN DEL DISEÑO

Algunas actividades del equipo de pruebas consisten en poder manipular algunos documentos que no necesariamente se involucran con el área de pruebas, como en el caso del documento de definición y especificación del diseño, sin embargo, estos documentos son necesarios para validar y garantizar que se respeten aspectos referentes al sistema a desarrollar para que éste sea lo que el cliente solicitó o que involucre todos los aspectos referentes a las reglas de negocios.

La estructura del documento consiste en identificar los diferentes componentes que conforman el *layout* o estructura visual del sistema, una vez identificados se realiza una estructura en tabla donde se les da un ID a cada componente, se determina qué tipo de componente es (si es una imagen, una tabla, un botón, texto) se le asigna un nombre corto, una breve descripción del campo, se define si el campo debe ser editable o fijo y si este contiene un valor por defecto.

Definición de Pantallas y Mensajes
V3.0.0 (01/12/2014)

Ámbito ☑ Aplicación	Nombre ☑ Bancomer móvil (Pago de tarjeta de crédito, Consulta de movimientos)		
Creado por ☑ Benjamín García Vásquez	Fecha Última Actualización ☑ 01/12/2014	Nº de versión ☑ 1.0	País ☑ México

1. DEFINICIÓN

Código: ☑ | Nombre: ☑ Pago de tarjeta de crédito (Consulta de movimientos) | Tipo de Ventana: | Componente Nuevo/Mod.: ☑

Descripción: ☑ Adición de importes para consulta de tarjeta de crédito

2. LAY-OUT

Información Contenida: ☑

ID	Nombre	Descripción del Campo	Nombre de la Etiqueta	Tipo de Control	Longitud	Editable (S/N)	Formato	Máscara de Captura / Despliegue	Valor x Defecto
1	Botón "Atrás"	El botón "Atrás" se mostrará dependiendo de la plataforma	btnAtras	Button		N			
2	Etiqueta "Titulo"	Se mostrará etiqueta con la leyenda "BBVA Bancomer"	lblTitulo	Label		N			BBVA Bancomer
3	Imagen "Consultar movimientos"	Se mostrará la imagen correspondiente a la consulta de movimientos	imgConsultarMovimientos	Image		N			
4	Etiqueta "Consultar movimientos"	Se mostrará etiqueta con la leyenda "Consultar | Movimientos"	lblConsultarMovimientos	Label		N			Consultar | Movimientos

3. ESPECIFICACIÓN

3.1 Validaciones y Dependencias entre Campos:

Código	ID Lay-Out	Forma de Validación	Dependencia

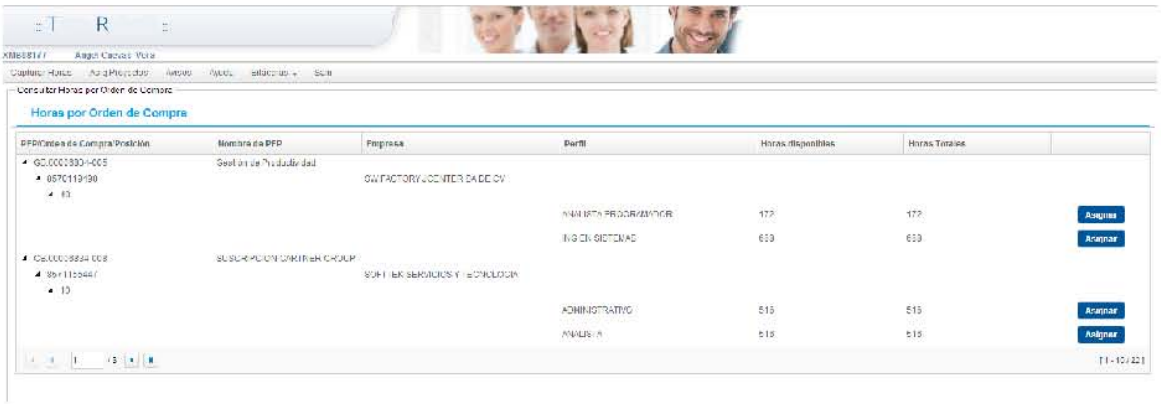
3.2 Acciones/Eventos/Funciones

Código	ID Lay-Out	Nombre del Evento	Descripción

3.3 Variables de la Ventana

Código	ID Lay-Out	Código Evento/Función Ventana	Ámbito	Tipo	Nombre

5. DISENO



viii) Fig. 4.1: Documento de definición y especificación del diseño

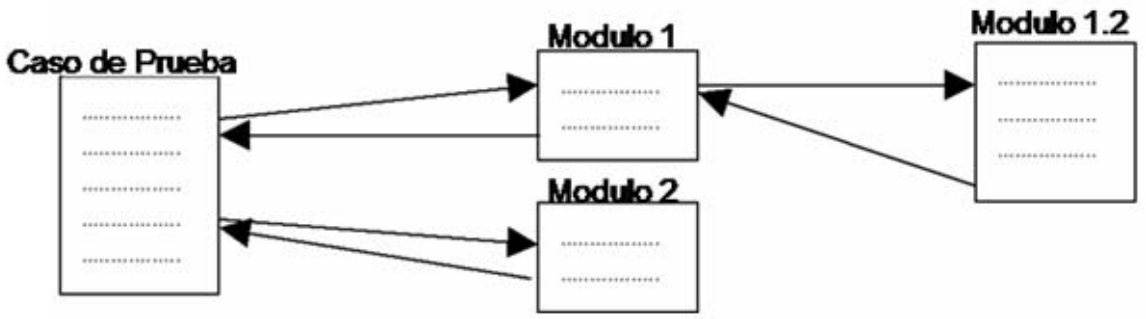
4.2 IMPLEMENTACION DE CASOS DE PRUEBA

En la ingeniería del software, los casos de prueba o *Test Case* son un conjunto de condiciones o variables bajo las cuales el analista determinará si el requisito de un sistema es completamente satisfactorio.

En esta sección se comentará sobre la creación del documento específico para reportar la implementación de casos de prueba, cuyo objetivo es describir las condiciones en las que se desarrollarán las pruebas, con el fin de validar la correcta funcionalidad de cada operación de su respectivo módulo dentro del sistema.

Un caso de prueba se puede definir como un escenario concreto de una prueba, es un documento utilizado para llevar un registro del análisis y diseño sobre los posibles escenarios que pueden presentarse para implementar casos de prueba orientados al sistema en desarrollo en el cual se deben contemplar las reglas de negocio definidas por el cliente, el camino ideal *Happy Path*, los escenarios funcionales, y *los* escenarios alternos.

Aquí se definen los modelos de la estructura de los casos de prueba.



ix) Fig. 4.2: Estructura casos de prueba

Cada módulo nos define una parte de todo un programa, algunos casos de prueba se pueden enfocar a uno o varios módulos. El trabajar con módulos facilita al equipo de desarrollo buscar y corregir las fallas que puedan presentarse en el sistema.

Los casos de prueba nos ayudan a validar que el sistema en desarrollo realice las funciones para las que ha sido creado con base a los requerimientos del cliente solicitante. Antes de desarrollar un caso de prueba es necesario saber cuál es el alcance y el objetivo del proyecto.

Al realizar el análisis sobre los posibles escenarios de los casos de prueba se debe llevar un registro de los resultados obtenidos al ejecutar los casos de pruebas, este documento es conocido como registro de observaciones, el cual se lleva a la par de la ejecución de los casos de prueba con el objetivo de evidenciar los diferentes resultados obtenidos, ya sean

negativos o positivos de las pruebas realizadas, al ejecutar los casos de prueba se considera trabajar con un control de versiones, esto es debido a que el cliente puede solicitar cambios al proyecto que afecte algún módulo o funcionalidad del sistema y este cambio puede afectar uno o varios casos de prueba, el control de versiones nos respalda en caso de que el cliente solicite regresar a los cambio originales de una versión.

También el llevar un control de versiones sirve al equipo de pruebas para tener un respaldo de los casos de prueba desarrollados con anterioridad, al realizar los casos de pruebas estos requerimientos se dividen por módulos, y por lo menos deberá existir un caso de prueba por cada requerimiento. De esta forma se podrá trabajar por requerimientos de una forma más organizada y controlada.

Este documento se conforma por el código de casos de pruebas, el cual es un identificador jerárquico usado para cada caso de prueba con la nomenclatura "CP" seguida de un número, empezando con el número "uno", en la categoría del caso se define si este va orientado al "*Happy Path*", escenario alterno, escenario de excepciones, escenarios funcionales, a las reglas de negocio, condiciones y datos de entrada, este último hace referencia a los datos que deben usarse para poder ejecutar este caso de pruebas, además se definen condiciones para que se pueda realizar, como sería el caso en el que se haya realizado un caso de prueba o un escenario anterior para poder realizar el escenario o caso de prueba actual.

Dentro de los pasos del caso de prueba, se definen un conjunto de instrucciones a seguir para poder ejecutar los casos de pruebas en el sistema, estos pasos se realizan bajo un escenario controlado donde se determinan los resultados esperados que debe arrojar el sistema, por lo cual estos pasos se realizan de forma ordenada y jerárquica, en los resultados esperados se hace referencia a la respuesta que dará el sistema para determinada acción dependiendo de los pasos del caso de prueba.

Estructura del documento para desarrollar los casos de pruebas.

	A	B	C	D	E	F	G
1							
2						Casos de Prueba	
3						*V1.0 (05/12/2014)*	
5							
6							
7	**Código Caso**	**Categoría del Caso**	**Condición y Datos de Entrada þ**	**Tipo Da Pa**	**Pasos del Caso de Prueba**	**Resultados esperados þ**	**Estatus**
8	**Escenario:** Usuario sin campaña Paperless.						
9				S	Abrir la aplicación Bmovil.		
10				S		La aplicación muestra la pantalla principal con las opciones *Banca Movil* y *Token Movil*.	
11				S	Seleccionar la opción *Banca Movil*.		
12				S		La aplicación muestra la pantalla *Acceso a Banca Movil*.	
13				V	Verificar que en el campo Número de Celular se muestre *****XXXXX del número de celular del usuario.		
14	CP01	Happy Path	Usuario debe estar dado de alta en la aplicación Bmovil	D	Ingresar contraseña XXXXX en el campo *Contraseña*.		Pendiente
15				S	Oprimir el botón *Entrar*.		
16				S		La aplicación bloquea la pantalla y muestra el Indicador de Actividad con el mensaje de Bienvenida.	
17				S	Oprimir el botón *OK* del Indicador de Actividad.		
18				S		La aplicación muestra la pantalla de *Menú Principal*, sistema no muestra pantalla emergente con *campaña paperless*.	
19				S	Usuario selecciona el boton *Home*.		
20				S		El sistema cierra aplicación *Bmovil*.	

x) Fig. 4.3: Documento para casos de pruebas

4.3 ESTRATEGIA Y PLAN DE PRUEBAS

En esta sección se comentará sobre la utilidad de la creación del documento estrategia y plan de pruebas, el cual brinda un soporte donde se define el objetivo, alcance y suposiciones para poder realizar las pruebas en el sistema, además de definir las restricciones y criterios de finalización de pruebas.

Para que se puedan empezar a realizar las pruebas en el sistema, éste debe tener todas o gran parte de las funciones solicitadas en los requerimientos para que el equipo de desarrollo entregue un producto que se apegue a lo solicitado por el cliente, los criterios de suspensión y reinicio de pruebas nos define el motivo por el cual las pruebas no se van a llevar acabo, ya sea porque en el sistema no se están respetando los requerimientos solicitados por el cliente, el sistema no hace lo que debería hacer o por que no se entregó una versión actual del sistema, en dicho documento se lleva la anotación detallando las actividades a realizar en cada caso de prueba, se define la dependencia de uno o varios casos anteriores para poder ejecutar un caso de prueba posterior, se lleva un registro de la fecha estimada y el periodo para realizar y finalizar las pruebas.

Se lleva una estimación de tiempos relacionados a los casos de pruebas por un registro de tiempos evaluados a las pruebas que se realizarán al sistema en el periodo definido por el cliente y después un registro del tiempo real en el periodo que se realizaron la pruebas.

Para esto se define una estimación en horas y minutos de en cuánto tiempo se deberían terminar las pruebas por cada caso, de forma independiente a los demás casos, se lleva un registro del inicio y finalización de las pruebas, se lleva a la par un registro si el caso de prueba fue exitoso, quedó pendiente, o no se pudo realizar por que no cumplió con algún requerimiento solicitado o definido por el cliente.

La importancia de este documento radica en que es la base para presentar un avance del proyecto en las fechas estimadas ante el líder de proyecto o en su debido caso al mismo cliente, y además para darle seguimiento a los casos de prueba que quedaron pendientes.

Este documento se finaliza cuando todos los casos de prueban son exitosos.

C109 - Estrategia y Plan de Pruebas BITÁCORAS(Alcance limitado)
v1.0 (9/12/2014)

Aplicación / Función	Aplicación				Nombre del proyecto	Emoviles						
Objetivo	El objetivo del plan de pruebas es verificar y validar que la campaña paperless y las mejoras a la Consulta de Estados de cuenta cubran los requerimientos planteados para dar solución a las necesidades identificadas, así como las debidas garantías de calidad, antes de su paso a producción.											
Creado por	Emanuel S/Emanuel H/Efraín C		Fecha Actualización	9-12-2014	Nº Versión	1	País	México				
Código de la Iniciativa												

						Fechas Estimadas		Fechas Reales				
ID	Actividad	Producto	P026 C. Uso	C204 C. Prueba	Plazo (Horas)	Fecha inicio	Fecha Fin	Fecha inicio	Fecha Fin	Dependencia	Responsable	Estado
0	Usuario con perfil recortado, no muestra campaña Paperless.	Aplicación permite acceso al usuario, no aparece el pop up de paperless por que usuario es perfil recortado		CP01	0.1							Pendiente
1	Usuario sin campaña Paperless	Aplicación permite acceso al usuario, no aparece el pop up de paperless		CP01	0.1							Pendiente
2	Usuario rechaza campaña paperless.	Aplicación permite acceso al usuario, aparece el pop up de paperless y el usuario lo rechaza. Se verifica con NACAR que el contador de rechazos aumente en uno		CP02	0.2							Pendiente
3	Usuario sin token activo acepta campaña paperless.	Aplicación permite acceso al usuario, aparece el pop up de paperless este acepta y tiene que activar el token, posteriormente acepta campaña paperless		CP03	0.2							Pendiente

xi) Fig. 4.4: Documento estrategia y plan de pruebas

4.4 REGISTRO DE OBSERVACIONES

En esta sección se comentará sobre la creación del documento específico registro de observaciones, debido a la alta probabilidad de errores o *bugs* que puedan ocurrir en el sistema, es necesario llevar un registro de las diferentes fallas que se presentan en el periodo de pruebas del sistema, para que posteriormente se dé una solución a estas fallas.

En el registro de observaciones se anotan las incidencias, fallas o errores ocurridos en el periodo de pruebas, el objetivo principal del registro de observaciones es para que el equipo de pruebas tenga las evidencias suficientes para presentarlas ante el equipo de desarrollo y estos tengan un mejor entendimiento de la incidencia.

Se utiliza un identificador por cada tipo de incidencia para enumerar de una manera organizada la identificación de los diferentes tipos de incidencias que se presentan en el periodo de pruebas, al levantar una incidencia por falla o error se realiza una captura de pantalla y se clasifican como incidencia, estas capturas nos sirven como evidencia de en qué momento se reflejó el error o la falla en el sistema, al realizar la captura de pantalla se incluye la fecha y la hora exacta de la incidencia, este método también sirve para justificar ante el equipo de pruebas las incidencia, escribiendo una corta pero detalla observación sobre la falla.

En algunas ocasiones el líder de proyecto solicitaba este documento para validar la cantidad de incidencias que se presentaron al terminar el periodo de pruebas, esto le servirá como avance del proyecto y para validar si algunas de estas incidencias ya se habían repetido o no se había solucionado en una entrega anterior.

Registro de Observaciones
V5.1 (03/05/2012)

Fecha de creación: 02-abr-14

Código Observación	Aplicativo	Código Caso de Prueba	Id Proceso	Nombre Proceso	Descripción de la observación	Evidencia	Tipo de Prueba	Tipo Observación	Severidad	Entorno	Estado	Responsable Prueba
01	Time Report	CP01	01	Identificación de Roles	El sistema no permite ingresar con un usuario interno y contraseña	NO USUA RIOS.png	Funcional	Incidencia	Crítica	Desarrollo	Diferida	Yoselin Pacheco
02	Time Report	CP01	01	Identificación de Roles	Jefe Inmediato no visualiza el menú Bitácoras	CP01.png	Funcional	Incidencia	Crítica	Desarrollo	Cerrada	Yoselin Pacheco
03	Time Report	CP02	01	Identificación de Roles	Responsable no visualiza el menú Bitácoras	CP02.png	Funcional	Incidencia	Crítica	Desarrollo	Cerrada	Yoselin Pacheco
04	Time Report	CP03	01	Identificación de Roles	Administrador TR no visualiza el menú Bitácoras	CP03.png	Funcional	Incidencia	Crítica	Desarrollo	Cerrada	Yoselin Pacheco
05	Time Report	CP04	01	Identificación de Roles	Director no visualiza el menú Bitácoras	CP04.png	Funcional	Incidencia	Crítica	Desarrollo	Cerrada	Yoselin Pacheco
06	Time Report	CP05	01	Identificación de Roles	Subdirector no visualiza el menú Bitácoras	CP05.png	Funcional	Incidencia	Crítica	Desarrollo	Cerrada	Yoselin Pacheco

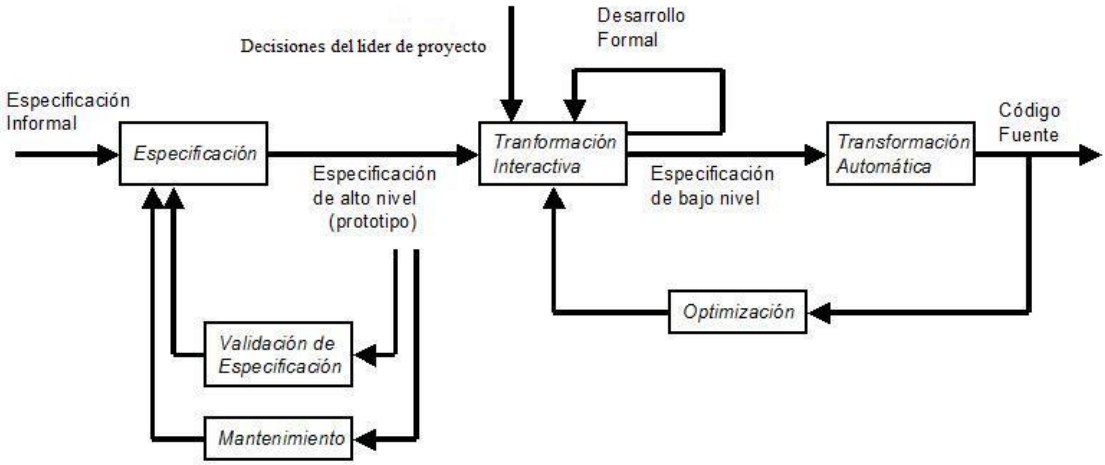
xii) Fig. 4.5: Documento registro de observaciones (parte 1)

Responsable Corrección de la observaci	Responsable de cierre	Fecha de Alta	Fecha de Asignación	Fecha Fin de Análisis	Fecha Real Disponible para validació	Fecha de cierre prevista	Fecha de cierre Real	Número de Iteraciones	*Periodo Corrección/Días abierta*	Origen de la observación	Comentarios	Compromisos (Responsable-Compromiso-Fecha)
Sistemas	Sistemas	02/04/2014	02/04/2014	02/04/2014	02/04/2014	16/05/2014			*0*	De Desarrollo	Dado que no se cuenta con un directorio Activo, las claves de usuarios Internos se introducen anteponiendo "X" y sin introducir contraseña	Validación para atender incidencia-Rogelio Illescas
Sistemas	Sistemas	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	1	*#NAME?*	De Desarrollo	Dado que no se cuenta con un directorio Activo, las claves de usuarios Internos se introducen anteponiendo "X" y sin introducir contraseña	Rogelio Illescas
Sistemas	Sistemas	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	1	*#NAME?*	De Desarrollo	Dado que no se cuenta con un directorio Activo, las claves de usuarios Internos se introducen anteponiendo "X" y sin introducir contraseña	Rogelio Illescas
Sistemas	Sistemas	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	1	*#NAME?*	De Desarrollo	Dado que no se cuenta con un directorio Activo, las claves de usuarios Internos se introducen anteponiendo "X" y sin introducir contraseña	Rogelio Illescas
Sistemas	Sistemas	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	1	*#NAME?*	De Desarrollo	Dado que no se cuenta con un directorio Activo, las claves de usuarios Internos se introducen anteponiendo "X" y sin introducir contraseña	Rogelio Illescas
Sistemas	Sistemas	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	02/04/2014	1	*#NAME?*	De Desarrollo	Dado que no se cuenta con un directorio Activo, las claves de usuarios Internos se introducen anteponiendo "X" y sin introducir contraseña	Rogelio Illescas

xiii) Fig. 4.6: Documento registro de observaciones (parte 2)

4.5 MODELO FORMAL DEL SISTEMA ORIENTADO A LA GESTIÓN DE PROCESOS EN TIME REPORT

Modelo donde entra en desarrollo las necesidades del cliente y se basa en los procesos específicos de los requisitos solicitados, con el fin de llegar a un sistema ejecutable.



xiv) Fig. 4.7: Modelo formal del sistema

Especificación: Es la acción de convertir un requisito informal del cliente a una especificación formal, para ser implementada en el sistema.

Validación de especificación: Es la acción de comprobar que la especificación informal se ajusta a la especificación formal.

Mantenimiento: Consiste en la conservación de la especificación formal para evitar su degradación.

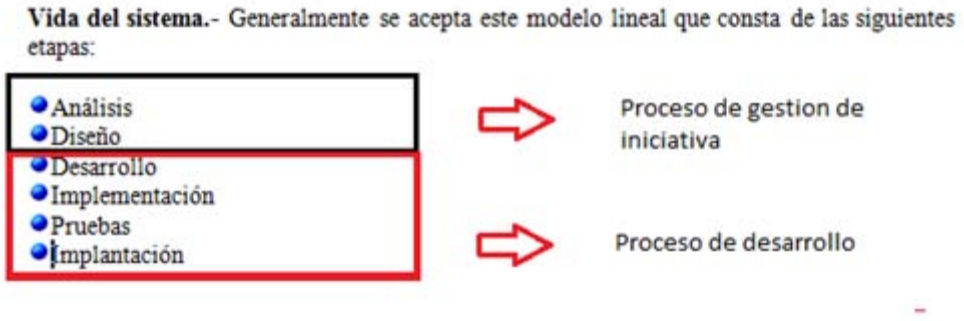
Transformación interactiva: Nos defina la relación entre el sistema y usuarios.

Transformación automática: Nos define la relación entre el sistema y el desarrollador.

Optimización: Es la acción de modificar el sistema para obtener mayor precisión respecto a la especificación formal

4.5.1 PROCESO DE GESTIÓN DE INICIATIVA Y PROCESO DE DESARROLLO

Consiste en un esquema lineal, donde cada módulo está orientado al modelo en cascada para tener una retro alimentación en el desarrollo o en la gestión de procesos, que sirven como referencia de las diferentes partes que componen un proyecto y su tiempo de vida.



xv) Fig. 4.8: Proceso de gestión de iniciativa y proceso de desarrollo

El proceso de gestión de iniciativa y desarrollo es una metodología corporativa de gestión, cuyo objetivo es mejorar el desempeño y la optimización de los procesos de negocio de la organización orientados al proyecto.

4.5.1.1 PROCESO DE GESTIÓN DE INICIATIVA

Los procesos de gestión de iniciativa consisten desde el análisis y la creación de un conjunto de metodologías para la estructura del proyecto.

En la parte de análisis se involucran: organización, documentación, estimaciones, costos, términos y condiciones.

En la parte de diseño se involucra: modelado, optimización, diseño general del proyecto y documentación.

4.5.1.2 PROCESO DE DESARROLLO

Consisten en procesos donde se involucra todo lo referente al desarrollo del proyecto y pruebas internas antes de la entrega e instalación con el cliente.

4.5.1.3 DESARROLLO BASADO EN REUTILIZACIÓN

Modelo utilizado por el equipo de desarrollo debido a que se va a trabajar con un sistema que ya fue implementado por otra fábrica de software, este modelo está fuertemente orientado a la reutilización:

1. Análisis de componentes: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes para adecuarlos.

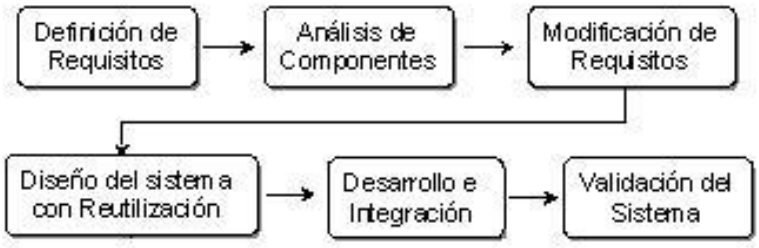
2. Modificación de requisitos: Se adaptan (en lo posible) los requisitos para concordar con los componentes de la etapa anterior. Si no se puede realizar modificaciones en los requisitos, hay que seguir buscando componentes más adecuados.

3. Diseño del sistema con reutilización: Se diseña o reutiliza el marco de trabajo para el sistema. Se deben tener en cuenta los componentes localizados en el segundo módulo para diseñar o determinar este marco.

4. Desarrollo e integración: El software que no puede comprarse, se desarrolla. Se integran los componentes y subsistemas. La integración es parte del desarrollo en lugar de una actividad separada.

Las ventajas de este modelo para el equipo de desarrollo:

- Disminuye el costo y esfuerzo de desarrollo.
- Reduce el tiempo de entrega.
- Disminuye los riesgos durante el desarrollo



xvi) Fig. 4.9: Modelo basado en reutilización

CAPÍTULO 5

METODOLOGÍA PARA LA EVALUACIÓN DE TIEMPOS DE ENTREGAS Y DESARROLLO DEL SISTEMA

Existen determinadas metodologías para definir los tiempos de evaluación del proyecto para las etapas de desarrollo, pruebas y entrega del proyecto, estos tiempos se valoran mediante una estimación de tiempos con base en la gestión de procesos del desarrollo del software por medio de la metodología PSP-TSP y las técnicas de ingeniería de software aplicadas a la programación (TISAP).

Un factor relevante para el desarrollo de un proyecto es el tiempo, esto es debido a que el cliente solo pagará las horas acordadas con la empresa, el líder del proyecto juega el papel más importante para la evaluación de tiempos debido a que es el intermediario entre el cliente, el equipo de desarrollo y el equipo de pruebas, una vez que el cliente define los requerimientos y el alcance del proyecto, el líder del proyecto junto con el líder del área de pruebas y el líder del área de desarrollo realizan una estimación de tiempos desde la producción, el desarrollo, las pruebas y la instalación del sistema, la cual es conocida como propuesta de tiempos, en la evaluación se toman tres conceptos importantes con base al tiempo, presupuesto y personas que colaboren a la realización del sistema, una vez que el líder de proyecto y el cliente evalúan la propuesta, el cliente puede aceptar la propuesta o en el caso de ser rechazada, se redefine y se realiza una nueva propuesta (esto puede suceder en caso de que el cliente no este conforme con los tiempos estimados debido a que estos puedan ser muy largos, o que se exceda del presupuesto estimado por el cliente), una vez que se redefinió la propuesta entre el cliente y el líder de proyecto, se propone el presupuesto, una vez que se acepta la propuesta, se da inicio la producción del sistema, después, el líder de proyecto realiza internamente una evaluación para contemplar la cantidad de personas que se deben involucrar o tendrán algún tipo de participación en el proyecto basándose en la estimación de tiempos y el presupuesto.

Es importante destacar que si por algún motivo el proyecto se atrasa por parte del equipo de desarrollo o de pruebas, cuando el cliente no solicitó alguna modificación o cambio en las ultimas etapas de desarrollo y el sistema ya debe de estar en la fase final de producción, pero el proyecto sigue atrasado, el cliente no pagará más de las horas y presupuesto acordados, y si este atraso se prolonga demasiado se puede llegar a convocar una auditoria para evaluar el proyecto y rendimiento de la empresa.

5.1 METODOLOGÍA PSP-TSP y FLEP-PT

Para llevar buenas prácticas de programación, optimizar tiempos y disminuir la cantidad de errores, es necesario tener una metodología que sirva como base para desarrollar un sistema eficiente y legible.
La metodología PSP – TSP desarrollada por Watts Humphrey en 1996 enfatiza las buenas prácticas individuales y grupales para la programación, orientándose a un proceso organizado en el desarrollo del software, esto con la idea de disminuir errores a la hora de desarrollar el código del sistema, estas metodologías se desarrollaron con la idea de asegurar la calidad del software.

5.1.1 PSP (PERSONAL SOFTWARE PROCESS)

PSP es un marco de trabajo de procesos que se implementa en la empresa para guiar a los desarrolladores y *testers* en diferentes aspectos, como:

- Definir sus propios procesos
- Planear y dar seguimiento a su propio trabajo
- Administrar la calidad de sus propios productos de trabajo

El PSP consiste en un proceso personal que, al estar basado en los principios de mejora, ayuda a los desarrolladores y a los *testers* a establecer sus metas personales, identificar qué métodos utilizarán para la realización del proyecto, medir su trabajo y analizar los resultados.

Para ajustar los métodos que utilizan para cumplir el objetivo del proyecto, este proceso ayuda a realizar mejor el trabajo, con el objetivo de obtener datos precisos y completos del proyecto, con el fin de mejorar el proceso individual, afectando de esta manera el desempeño de todo el equipo.

EL PSP se divide en 3 secciones (PSP0, PSP1 y PSP2), cada sección nos define:

PSP0 - Base de Ejecución Medible, Registro de Tiempos y Defectos.

Esta sección tiene la finalidad de documentar el plan del sistema a construir, además de que se realizan varias actividades como: diseño, codificación y compilación

PSP1 - Estimación de Tamaños, Esfuerzo y Plan.

Estima el tamaño del sistema a construir y se preparan los casos de prueba

PSP2 – Calidad, Prácticas de Revisión e Inspección.

Se realiza la revisión del diseño y la revisión del código, además de la prevención y eliminación de defectos.

5.1.2 TSP (TEAM SOFTWARE PROCESS)

Este modelo sirve para proveer un énfasis en los procesos, los productos y el trabajo en equipo tomando de base los principios de PSP, pero con algunas diferencias, para realizar los procesos y principios de ingeniería de software en un ambiente de trabajo en equipo.

TSP se enfatiza en que:

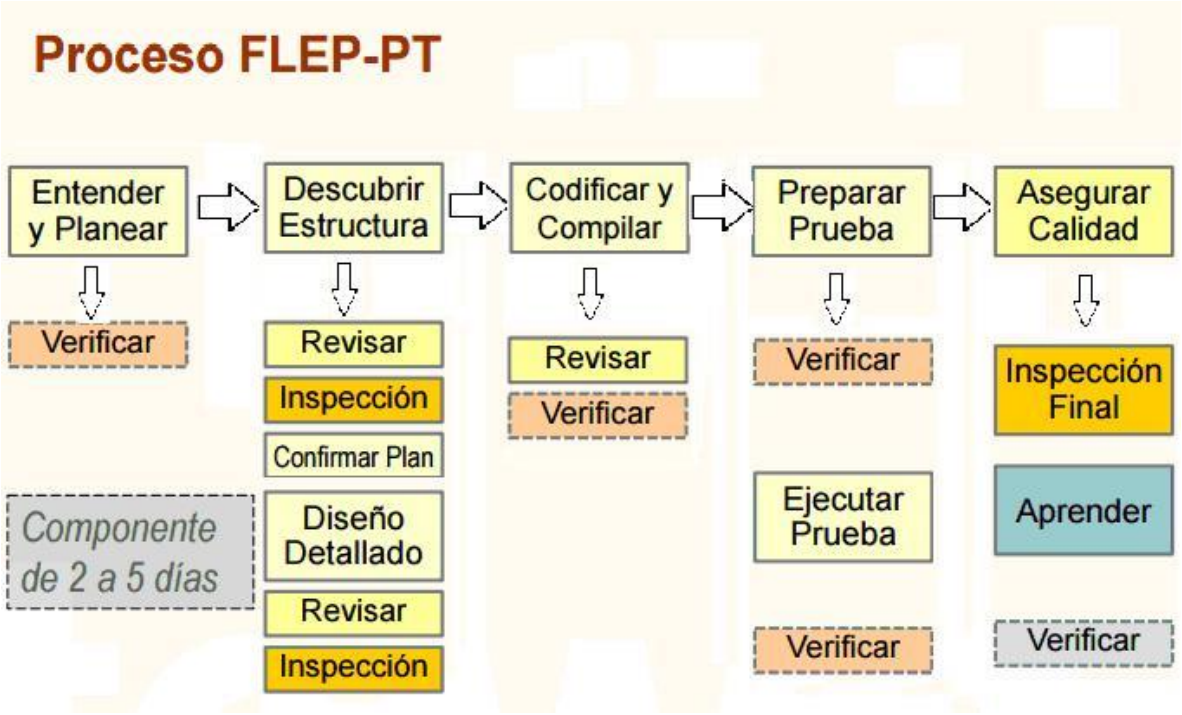
- Los equipos no se forman mágicamente, (el equipo de trabajo es seleccionado por sus diferentes cualidades orientadas al desarrollo del proyecto).

- Se deben entender las fortalezas/debilidades de cada miembro del equipo y cómo estas soportan el desempeño del mismo. (Cada individuo depende de otras personas del equipo para realizar el trabajo en áreas que no tienen experiencia para el desarrollo del proyecto).

- Se requiere una estrategia definida para trabajar juntos de manera coordinada, establecer responsabilidades y dar seguimiento al avance. Esto se logra teniendo metas comunes, acordando planes de acción y con un liderazgo apropiado.

5.1.3 DESCRIPCIÓN DE FASES DEL PROCESO FLEP-PT

FLEP-PT (ejecución impecable de las tareas de programación) es un proceso desarrollado por la empresa TOWA conformado por fases orientadas a un esquema estructurado para el desarrollo del software basado en las metodologías PSP y TSP, este esquema nos define cinco principales fases de buenas prácticas orientadas a disminuir errores en la etapa de desarrollo para garantizar una mayor calidad en el desarrollo del sistema, mediante un registro que se define desde la etapa inicial de entendimiento y planeación del proyecto hasta la etapa inspección y calidad.



xvii) Fig. 5.1: Estructura del proceso FLEP-PT

Descripción de los registros de las fases que conforman la estructura del proceso FLEP-PT:

Componente. - El término de componente hace referencia a un módulo o una parte del proyecto a realizar por el equipo de desarrollo, también nos define el tiempo ideal que es de 2 a 5 días para que concluyan el componente o módulo del sistema.

Entender y planear. - Leer cuidadosamente y entender el detalle de todos los elementos de la especificación (se explica el "qué" se debe hacer) del componente a desarrollar (documentos de base, documentos anexos, documentación de código pre construido por el diseñador, documentación de código de uso, elementos de la arquitectura que se requieran, etc.) en el nivel de detalle que se necesite.

Con este entendimiento, el desarrollador hace un primer estimado del tamaño y del esfuerzo que le llevará el trabajo y lo registra en su plan. Aquí el desarrollador estima las posibles líneas de código que le llevará realizar en el sistema por módulo y el tiempo que tardará.

Descubrir Estructura. - Al tener un entendimiento de lo solicitado por el cliente, el desarrollador hace un diseño general que refleje la estructura del sistema a desarrollar, por medio de un diagrama de flujo orientado a la solución del sistema.

En esta parte, se deben enfocar a identificar las estructuras de procesamiento básicas de las partes del problema.

Revisar Estructura. - El desarrollador debe revisar su propio trabajo antes de pedir una inspección por parte del líder del proyecto o en determinados casos se revisará con un superior.

Inspeccionar Estructura. - Cuando el líder de proyecto o superior revisa la estructura, este decide si esta es correcta o si se debe rediseñar, en esta parte también se define si se debe optimizar la estructura y se remueven algunas partes que no se consideren necesarias.

Confirmar Plan. - Una vez que se termina la fase de inspección de la estructura, se re-estima el tamaño y esfuerzo del sistema, Los estimados en este paso son los definitivos y éste será el compromiso con el proyecto para obtener el producto final (como se definió en cuanto a tiempo, calidad y costo).

Diseño Detallado. - Se define claramente el objetivo detallado de cada parte o módulo que constituye el sistema, así como sus parámetros en cuanto a cantidad y tipo.

Se deben considerar todos los detalles del sistema, visualizar de forma detallada todas las posibilidades de la función de cada parte con un enfoque de solución local, elegir la opción óptima y su interacción con las otras partes para resolver el problema, también el desarrollador debe enfocarse a identificar las estructuras de procesamiento básicas de las partes del problema, si es necesario se realiza una prueba de escritorio, esto es, con base a los casos posibles que se identificaron en el detalle para cada una de las partes.

Revisar Diseño Detallado. - En caso de que exista alguna duda sobre el proyecto, se revalúan los elementos de las especificaciones y se resuelven dudas o problema referentes al sistema. Si se encuentran defectos, se deben remover y se realizan los ajustes necesarios.

Inspeccionar Diseño Detallado. - Una vez que el diseño detallado es correcto, este es inspeccionado por el líder de proyecto o un superior.

Codificar y Compilar. - Se elabora la codificación de cada parte apegándose por completo a la última versión del documento de estándares de código. Se realiza un mapeo del diseño detallado a las instrucciones del lenguaje que se usa, se resuelven los problemas de codificación en el momento y se aseguran que se pueda compilar cada parte antes de seguir avanzando.

Revisar Codificación y Compilación. - Se revisa cuidadosamente la codificación asegurando que se apegue por completo al diseño detallado en cuanto a la lógica y a los estándares del lenguaje que usa.

Verificar Codificación y Compilación. - Con el objetivo de asegurar una buena codificación y compilación de cada componente, el líder del proyecto o un superior debe hacer preguntas al desarrollador enfocadas a confirmar la codificación y la compilación del sistema.

Preparar Prueba. - A partir de esta fase se declara la terminación de la programación, la cual queda suspendida hasta terminar el periodo de pruebas.

Con el entendimiento del problema y su solución, se plantean las diferentes condiciones del diseño detallado de cada parte en los casos de prueba, viendo el problema como un todo.

Se deben considerar escenarios diferentes para los casos de prueba (distintas situaciones que puedan ocurrir) para obtener una alta calidad en el producto final. Se deben preparar los casos de prueba con datos, de forma ordenada según la estructura.

Verificar Preparar Prueba. - Con el objetivo de asegurar una buena elaboración de casos de prueba, el *tester* debe hacer preguntas al desarrollador enfocadas a confirmar la verificación del sistema en desarrollo.

Si se encuentran defectos se deben registrar, y realizar ajustes necesarios a los casos de pruebas.

Ejecutar Prueba. - Con los casos de prueba preparados, y la última actualización del código fuente, se realizan las iteraciones necesarias de la ejecución del sistema.

De acuerdo a los resultados esperados para cada caso de prueba, se registra si el resultado proporcionado cumplió con el resultado esperado o no, y se documenta lo necesario en el listado del sistema.

Las pruebas se terminan cuando todos los resultados esperados en los casos de prueba se cumplen con los resultados proporcionados, haciendo las iteraciones, registros y ajustes necesarios.

Verificar Ejecutar Prueba. - Con el objetivo de asegurar una buena ejecución de los casos de prueba, el líder de proyecto o superior debe hacer preguntas al desarrollador enfocadas a confirmar la ejecución del sistema.

Asegurar Calidad. - Una vez terminado el programa y antes de entregar los resultados, se revisan las especificaciones con el objetivo de asegurar la calidad del producto.

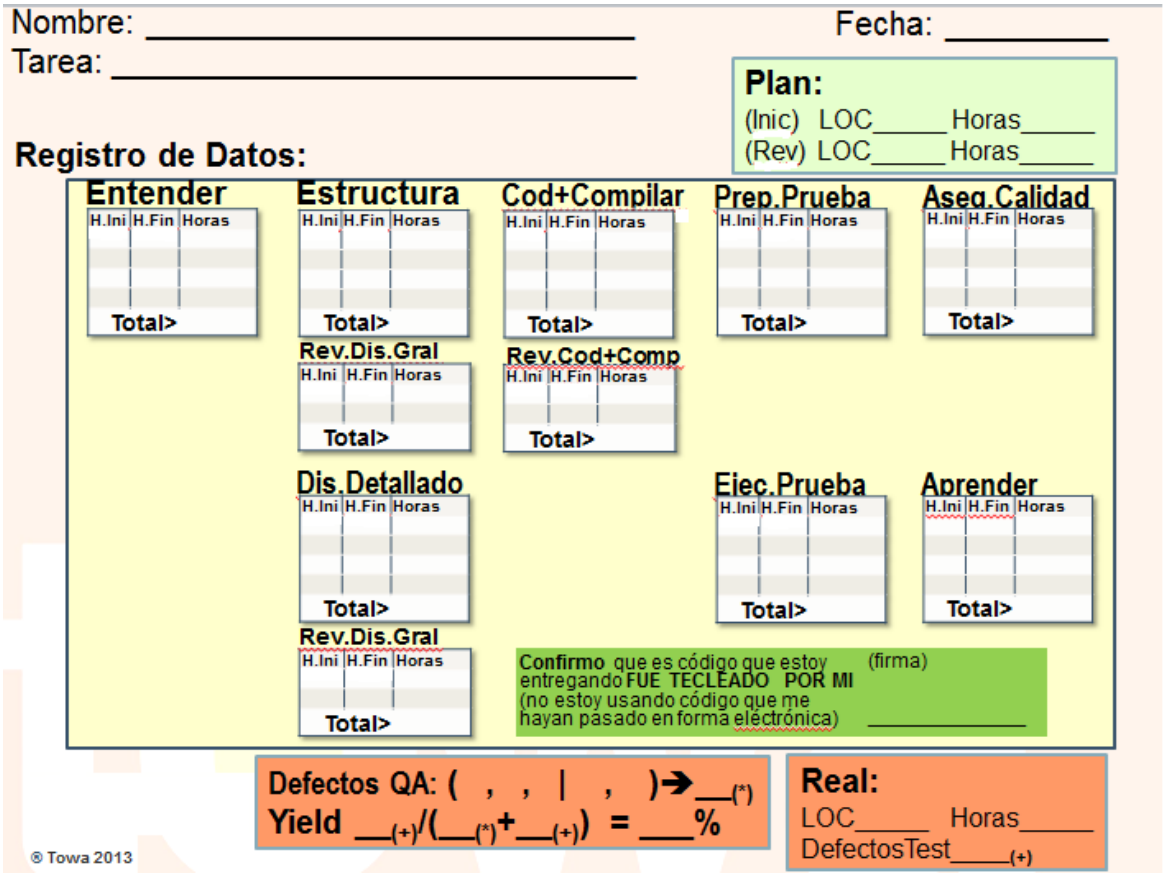
Inspección Final del Programa. - La inspección la realiza el líder de proyecto antes de la entrega con el cliente.

5.2 HOJA DE REGISTRO DE DATOS

Es el método por el cual se procede a registrar las fases del FLEP, la cual sirve para anotar los respectivos tiempos de las horas invertidas para el desarrollo estructural del sistema y también sirve para el registro de las líneas de código que se deben implementar en el sistema.

5.2.1 IMPLEMENTACIÓN DEL REGISTRO DE DATOS

En el siguiente diagrama se visualizan los diferentes elementos que conforman cada componente de la hoja de registro de datos, así como su funcionalidad. Posteriormente se describen cada uno.



xviii) Fig. 5.2: Hoja de registro de datos del proceso FLEP-PT

Nombre. - Implica el nombre o nombres de los desarrolladores involucrados para realizar el sistema, ya sea una especificación, tarea, módulo o sub-módulo en específico.

Tarea. - Hace referencia ya sea a todo el sistema en desarrollo, a un módulo, o un sub-módulo, que parten como una pequeña parte del sistema.

Plan. - Etiqueta donde se involucra el primer registro que parte de una estimación realizada por el desarrollador en conjunto con un superior o con el líder de proyecto, en la cual se estima el tiempo y las líneas de código (*LOC*) que tardarán en desarrollar las fases del sistema, esto sirve para determinar si realmente el tiempo y las líneas de código que se estimaron por el desarrollador se apegan a los tiempos y líneas de código reales.

Inc.- Etapa inicial donde se estiman las horas y líneas de código que deben invertirse para el desarrollo el sistema.

Rev.-Es la etapa de revisión, donde el superior determina las horas y las líneas de código (*LOC*) que él considera que debería tardar en desarrollarse la tarea.

En las siguientes fases que se comprenden desde entender hasta aprender, se anotarán los tiempos reales correspondientes a la realización de las determinadas actividades de cada estructura.

Estas estructuras están formadas por los siguientes elementos:

H. Ini. - Es la hora de inicio de la actividad de cada fase, la cual se registrará como hora y minutos (HH:MM).

Fin. - Hora de fin de la actividad, ya sea por alguna interrupción o al terminar la actividad de una fase y se registra como hora y minutos como la hora de inicio (HH: MM).

HORAS. - Tiempo entre hora inicio y la hora fin de dedicación a la actividad en proporción a la hora, se calcula como:

$$\textbf{HORAS} = \frac{((\text{H.Fin} - \text{H.Ini}) * 100)}{60}$$

Al obtener el resultado de la operación, si es el caso que salgan números decimales, estos se redondean.

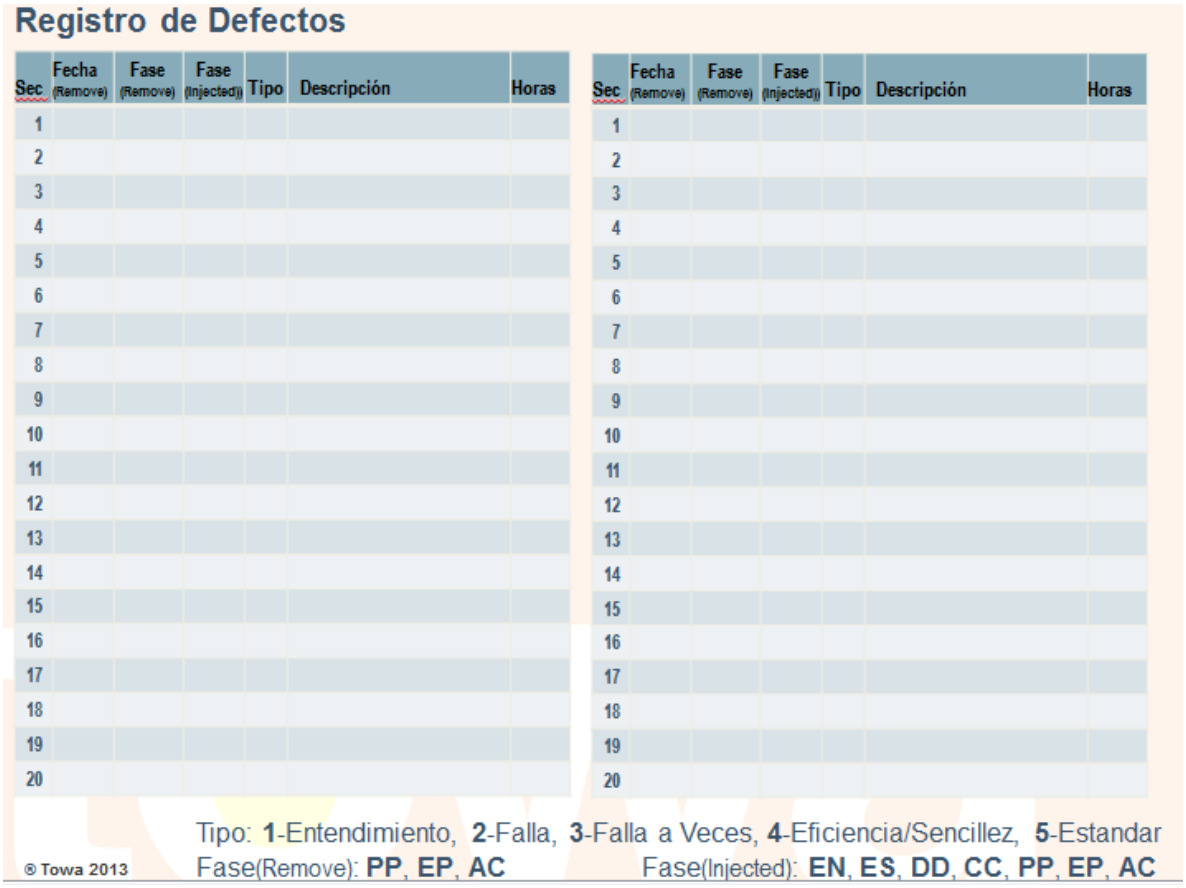
TOTAL. - Tiempo total de la columna horas del respectivo bloque, este se debe calcular para todos los bloques de todas las fases.

REAL HORAS. - Al terminar el desarrollo del sistema, se debe obtener el tiempo total real que se le dedicó, para esto se deben sumar todos los totales de los bloques de las diferentes fases.

Defectos de Test. - Cantidad de defectos encontrados y removidos por el desarrollador en la fase de ejecución de pruebas. Se calcula como la suma de defectos tipo 1 (entendimiento), 2 (falla) y 3 (falla a veces). Los cuales se desglosan en el registro de defectos.

5.2.2 REGISTRO DE DEFECTOS

En esta fase como se muestra en la figura 5.3, se utiliza una tabla donde se registran los defectos que se remueven cuando se revisan las estructuras del diseño de las pruebas y de calidad.



xix) Fig. 5.3: Hoja de registro de defectos del proceso FLEP-TP

La tabla del registro de defectos sirve para llevar una anotación de los diferentes registros que se van manifestando en el sistema, los cuales son anotados con sus respectivas características, este registro está conformado por:

Sec.- Número del defecto. Tiene la función ser un contador para enumerar los defectos de forma organizada.

Fecha (Remove). - Fecha en la que se remueve el defecto encontrado.

Fase de Remoción. - Fase donde se remueven defectos encontrados, en esta fase se involucran 3 estructuras del registro de defecto:

- PP.- Preparar prueba
- EP. - Ejecutar prueba
- AC. - Asegurar Calidad

Fase de inyección (Injected). - Fase en la que se inyectaron los defectos, esta fase involucra siete estructuras de registro de defectos:

- EN.- Entender y planear
- ES. - Descubrir estructura
- DD.- Diseño detallado
- CC.- Codificar y compilar
- PP.- Preparar prueba
- EP. - Ejecutar prueba
- Tipo. - Tipo del defecto inyectado (Estándar de la empresa)

YIELD. - Rendimiento del proceso, es el porcentaje de defectos inyectados y removidos antes de la ejecución.

$$\textbf{YIELD} = \frac{100 - \text{Fase (Remove)}}{\text{Fase (Injected)}}$$

Descripción. - Causa raíz que ocasionó el defecto (no solamente es el síntoma).

Horas. - Tiempo de remoción del defecto.

Existen cinco diferentes tipos de defectos los cuales se pueden clasificar como críticos y no críticos:

CRÍTICOS

T1.- Tipo de defecto 1 (Entendimiento)

Cualquier defecto que se asocie con el entendimiento del módulo o del sistema, esto es en el caso en el que el desarrollador este realizando su respectiva tarea, pero este todavía no haya comprendido la parte a realizar.

T2.- Tipo de defecto 2 (Falla).

Cualquier error que se encuentre en el código, el cual puede de ser lógico o de sintaxis, y que provoque un mal funcionamiento del sistema.

T3.- Tipo de defecto 3 (Falla a veces).

Cuando el error solo se manifiesta en determinado momento, y este no siempre se sucede.

NO CRÍTICOS

T4.- Tipo de defecto 4 (Eficiencia/Sencillez).

Cuando el defecto es ocasionado por una falla o problema relativamente sencilla y que no requiere más de una hora corregir el defecto.

T5.- Tipo de defecto 5 (Estándar).
Cuando no cumple con alguna política, regla de negoció o estándares del sistema de desarrollo.

Defectos QA: Cantidad de defectos encontrados y removidos en la revisión de la fase de calidad. Es la suma de los defectos críticos tipo T1, T2 y T3, no se consideran los defectos T4 y T5**.**

Defectos QA: (T1, T2, T3| T4, T5)

CAPÍTULO 6

PERIODO DE PRUEBAS

Al iniciar la fecha definida por el cliente y el líder de proyecto en la que se debe empezar el periodo de pruebas y la fecha en la que el equipo comenzó a realizarlas, no existe una estimación del periodo de prueba real por cada uno de los casos de prueba que se definieron, ya que a nivel de producción puede existir un atraso en la liberación del sistema por parte del equipo de desarrollo, o por otro lado que se solicite un cambio o un nuevo requerimiento por parte del cliente, sin embargo aunque el periodo de pruebas se atrase, siempre debe estar apegado a los tiempos reales, o al menos a la fecha final para terminar el periodo de pruebas.

Una vez finalizado el periodo de pruebas y todos los escenarios en el caso de pruebas son exitosos, se realiza la instalación del sistema con el cliente y se define una fecha para realizar pruebas con el cliente con información o datos reales.

6.1 PRUEBAS REGRESIVAS DEL SISTEMA E IMPLEMENTACIÓN DE MEJORAS

Se denominan pruebas de regresión a cualquier tipo de pruebas de software que intentan descubrir errores (*bugs*), carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, causado por la realización de un cambio en el programa.

Las pruebas de regresión se realizan después de la corrección de un defecto o después de la adición de nuevas funcionalidades. Su objetivo es asegurar que ningún defecto se añadió al sistema después de la modificación y que las correcciones sean funcionales.

Si se llega a realizar un cambio en el desarrollo del sistema por petición del cliente, una vez que se probó un requerimiento del sistema y este resultó ser funcional, se debe probar de nuevo el mismo módulo para demostrar que el cambio a un módulo independiente o dependiente, no afecte de forma negativa a algún requerimiento que ya se había probado.

6.1.1 METODOLOGÍA USADA

Una vez que se ejecutaron los casos de prueba de uno o varios requerimientos, se clasifican las capturas de pantallas por éxitos o incidencias. Una vez terminados los casos de pruebas, estas capturas exitosas se incorporan a los casos de pruebas que se reportan satisfactorios, y las capturas de las incidencias se incorporan al registro de observaciones, después se solicita una junta con el líder de desarrollo o con el equipo de desarrollo, y en ocasiones también al líder de proyecto cuando este solicita un avance del sistema, una vez que se aclaran las incidencias y los requerimientos exitosos, se regresa el sistema para su corrección y modificación por parte del equipo de desarrollo, después de que estas incidencias son corregidas, se reinicia el periodo de pruebas, y se vuelve a probar los requerimientos donde había incidencias, además los requerimientos que ya habían sido exitosos se vuelven a validar para confirmar que sigan estando bien.

6.2 CRITERIOS DE FINALIZACIÓN

Cuando el área de desarrollo libera uno o varios módulos del proyecto, este puede dar luz verde al área de pruebas para que puedan ejecutar sus casos de prueba previamente diseñados sobre el sistema.

6.2.1 SUSPENSIÓN Y REINICIO DE PRUEBAS

El periodo de pruebas se puede suspender y reiniciar en determinados casos:

1.- No se cuenta con el total de funcionalidades necesarias para realizar las pruebas. Estos casos pueden ser tanto por parte del equipo de desarrollo que tengan un módulo incompleto o la base de datos no está actualizada con datos reales, y en el caso del área de pruebas que no se tengan todos los casos de prueba de al menos un módulo que se desea probar y este caso de prueba tenga dependencia de otro.

2.- No se cuenta con la información necesaria y correcta del sistema. Cuando no contamos con suficiente conocimiento de los requerimientos o estos no estén completamente definidos por el cliente.

3.- La carga de ventanas o *layouts* no se ejecutan o no lo hacen de forma correcta.

4.- Cuando al realizar las pruebas, los *layouts* muestran un comportamiento no deseado esto se levanta como incidencia y se regresa el sistema a desarrollo.

5.- Las pruebas regresivas no están arrojando los resultados esperados. Una vez que un módulo está probado de forma correcta, se realizan pruebas sobre otro módulo, y el módulo que con anterioridad era correcto empieza a tener errores.

6.- Errores críticos detectados.

6.2.2 CUMPLIMIENTO DE LOS REQUERIMIENTOS PARA LA FINALIZACIÓN DE PRUEBAS

Los criterios de finalización de pruebas se cumplen cuando se realizan los requerimientos postulados por el cliente, y las características del sistema que se desarrollaron, los cuales están definidos en el documento E200-Business case.

Las pruebas finalizan cuando se llevó a cabo el objetivo principal y cada uno de los módulos de los requerimientos del sistema sea funcional.

En esta parte, ya el sistema se considera libre de errores, y se cuenta con el registro de los documentos: estrategia y plan de prueba y el registro de observaciones finalizados.

A pesar de que a partir de este punto el sistema no debería presentar ningún error, pueden presentarse diferentes fallas cuando el sistema ya se encuentra en etapa de producción debido a diversos factores que no se contemplaron en los casos de pruebas y estos pueden afectar al sistema.

CAPÍTULO 7

DISEÑO Y DESARROLLO DE LA INFRAESTRUCTURA DE PRUEBAS

7.1 ANTECEDENTES

Cem Kaner define el *testing* como una investigación técnica de un producto bajo prueba con el fin de brindar información relativa a la calidad del software, a los diferentes actores involucrados en un proyecto.

A partir de la información obtenida del *testing* se pueden tomar decisiones. Las decisiones pueden ser desde cuándo liberar un producto a producción, conociendo los riesgos que esto implica, hasta cómo mejorar las diferentes áreas dentro de la empresa. En definitiva, el *testing* es un agente de cambio, lo importante es interpretar la información obtenida para que todos los actores puedan actuar en forma oportuna donde sea necesario.

En el software la confianza es un elemento importante ya que ciertas fallas pueden tener consecuencias indeseables como pérdidas de dinero, negocios e incluso vidas, dependiendo de qué tan crítico sea el dominio en el cual el software interactúa. Las pruebas le dan valor agregado a cada proyecto brindando confianza a los distintos actores, el *testing* es una actividad cognitiva y no mecánica ni repetitiva que involucra varias funciones mentales como el lenguaje, la imaginación, percepción, entre otros.

Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo.

Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno corresponde un nivel distinto de compromiso en las actividades de desarrollo, los diferentes tipos pruebas se pueden clasificar ya sea por sus niveles (pruebas unitarias, pruebas de integración, pruebas de sistema, pruebas de aceptación), por su funcionalidad (pruebas de humo, pruebas de regresión, pruebas de aceptación, pruebas funcionales), por su enfoque (pruebas de caja negro, pruebas de caja blanca) y por su ejecución (pruebas manuales, pruebas automáticas).

A pesar de que existen una gran variedad de metodologías, técnicas, enfoques y formas para realizar pruebas en un sistema, un *tester* debe saber seleccionar y utilizar las pruebas que mejor se adapten a las necesidades del cliente y al sistema.

7.2 SOFTWARE DE PRUEBAS

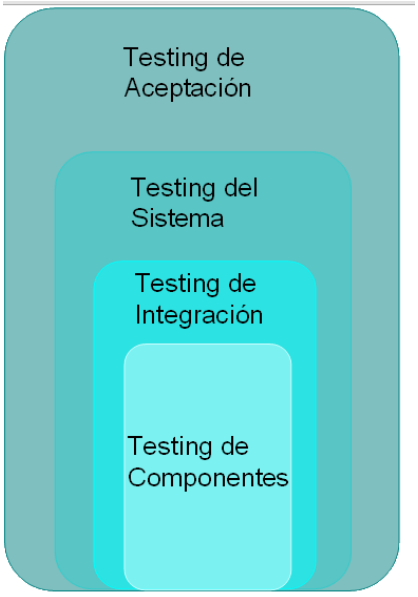
Las pruebas de software son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o *stakeholder*. Es una actividad más en el proceso de control de calidad.

Existen muchas y diversas formas en las que un sistema informático podría llegar a presentar fallas en su operación, algunas ocasionadas por distracción del mismo desarrollador o programadores, otras tantas por motivos de un cambio de equipo de trabajo a la mitad del desarrollo de un proyecto grande; cualquier que sea la razón existen sistemas que son más vulnerables a estas fallas por la complejidad que conllevan y es que para la comprensión de todo un proyecto como el que representa este trabajo se involucran varios factores, y como diría la ley de Murphy: “si algo puede salir mal, saldrá mal”, un sistema no puede ser perfecto, por que el ser humano no es perfecto, el *tester* debe tener la capacidad de entender en su totalidad el alcance del sistema, para que este pueda identificar todas las fallas o cuestiones que tengan un comportamiento diferente al que debería tener, además debe de encontrar un mecanismo efectivo que le permita corroborar el correcto funcionamiento del sistema y además que tenga un equilibrio entre calidad y cantidad de casos de prueba.

Los defectos de software siempre van a existir, por lo que se debe contar con herramientas que permitan determinar por medio de cifras numéricas veraces, si un programa o sistema funciona correctamente.

7.2.1 NIVELES DE PRUEBAS

El siguiente modelo representa los diferentes niveles de *testing*, el diagrama enfatiza la correspondencia entre *testing* a nivel estructural.



xx) Fig. 7.1: Niveles de *pruebas*

7.2.2 NIVEL DE PRUEBA DE COMPONENTES

También denominadas pruebas de desarrollo (*developer´s test*), se prueba cada componente tras su realización/construcción. Dadas las convenciones de cada lenguaje de programación para la asignación de nombres a sus respectivos componentes.

Su alcance se define como:

1) Probar componentes individuales.

2) Probar componente de forma independiente.

7.2.3 NIVEL DE PRUEBA DE INTEGRACÍON

Verifica las interfaces entre componentes, las interacciones entre las diferentes partes de un sistema, tales como el sistema operativo, sistema de archivos, *hardware*, o interfaces con otros sistemas.

Puede haber más de un nivel de pruebas de integración y puede llevarse a cabo sobre objetos de prueba de tamaño variable.

Su alcance se define como:

- Probar grupos de componentes.
- Comprobar la interacción entre componentes respecto de la especificación de interfaces.

7.2.4 NIVEL DE PRUEBA DEL SISTEMA

La calidad del software es observada desde el punto de vista del usuario. El entorno de prueba debe corresponder y parecerse al entorno en producción tanto como sea posible.

Su alcance se define como:

1) Adecuación ¿Las funciones implementadas son adecuadas para su uso esperado?
2) Exactitud ¿Las funciones presentan los resultados correctos?
3) Interoperabilidad ¿Las interacciones con el entorno del sistema presentan algún problema?
4) Cumplimiento de funcionalidad ¿El sistema cumple con normas y reglamentos aplicables?
5) Seguridad ¿Están protegidos los datos/programas contra acceso no deseado o pérdida?

7.2.5 NIVEL DE PRUEBAS DE ACEPTACIÓN

Las pruebas de aceptación son a menudo responsabilidad de los clientes y/o usuarios del sistema. La meta en las pruebas de aceptación es establecer confianza en el sistema, las partes del sistema o las características específicas y no funcionales del sistema.

Encontrar defectos no es el foco principal en las pruebas de aceptación. Las pruebas de aceptación pueden evaluar la disposición del sistema para el uso, aunque no es necesariamente el nivel final de las pruebas.

Su alcance se define como:

Verificar que el software satisface los requerimientos del cliente.

7.2.6 PRUEBAS UNITARIAS

Se evalúa el comportamiento de un segmento o módulo de código, el cual puede ser una función, un ciclo, o un condicional, esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado, esto nos facilita que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores nuevos en el sistema. Es importante considerar que las pruebas

unitarias no descubrirán todos los errores del código es necesario complementarse con otros tipos de pruebas.

7.2.7 PRUEBAS DE INTEGRACIÓN

Se realizan en el ámbito del desarrollo de software por parte del equipo de desarrollo, una vez que se han aprobado las pruebas unitarias, se deben integrar los módulos independientes al sistema, una vez realizada la integración se prueban todos los elementos unitarios que componen un proceso para hacer que funcionen conjuntamente para verificar el comportamiento de todos los módulos de forma unificada, la complejidad radica en unir los diferentes módulos, ya que estos pueden comportarse de forma irregular y arrojar resultados diferentes a los esperados.

7.2.8 PRUEBAS DE SISTEMA

Conforma diversas pruebas de software conjuntamente como pruebas de seguridad, recuperación, desempeño, disponibilidad, integridad de datos y almacenamiento, es similar a la beta *testing*, ya que el equipo de pruebas analiza y prueba el comportamiento del sistema desde la perspectiva de un usuario.

7.2.9 PRUEBAS DE ACEPTACIÓN

Estas pruebas se realizan para que el cliente certifique que el sistema es válido y aceptable, son básicamente pruebas funcionales sobre el sistema completo. Se busca comprobar que se satisficieron los requerimientos establecidos y su ejecución es dependiente de los criterios del cliente y en caso de que no se realicen todos los casos de prueba de forma explícita se dan por incluidos dentro de las pruebas funcionales del cliente, es decir las pruebas de aceptación serán responsabilidad del cliente o usuario, donde se representa un entorno de producción con datos reales e información real del sistema, el cliente cuenta con el soporte, apoyo y orientación de un reducido grupo selecto del área de desarrollo y del área de pruebas para que puedan realizar las pruebas que el usuario desee hacer para probar el sistema.

7.3 PRUEBA FUNCIONAL

El servicio de *testing* funcional permite a las empresas determinar si han construido el software deseado y si es oportuno liberar la versión del producto al mercado.
También permite a las empresas, que hacen uso intensivo de las tecnologías de la información, determinar si han adquirido el software deseado o si es oportuno aceptar la versión del producto liberado por su proveedor.

Este análisis se hace en común acuerdo con el cliente, quien identifica la lista priorizada de funcionalidades a probar, considerando su complejidad y criticidad. Una vez definido el alcance de las pruebas, se planifica y define la estrategia a seguir.
Se propone una estrategia basada en el análisis de riesgos del producto, definiendo claramente el contexto y los objetivos.

7.3.1 PRUEBAS DE FUNCIONAMIENTO

Las pruebas de funcionamiento permiten conocer y reducir los riesgos relacionados con el mal desempeño de las aplicaciones en los entornos de producción y realizar las correcciones necesarias antes de salir al mercado. Aquí también se involucran pruebas de estrés, volumen de datos y cargas.

Se cuantifica la capacidad de la infraestructura, se validan los requerimientos de la escalabilidad de las plataformas y del sistema a probar.

De esta manera, la empresa puede conocer qué cantidad de clientes simultáneos soporta su producto, con tiempos y datos razonables sobre la infraestructura y las plataformas propuestas.

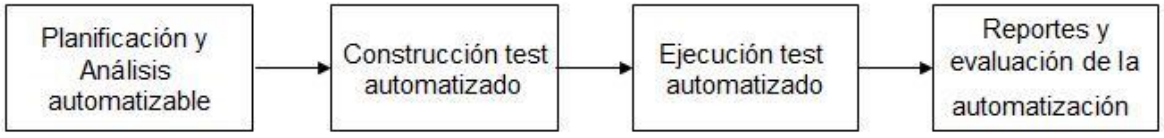
Asimismo, puede saber si es suficiente el hardware para soportar el nivel propuesto de transacciones y qué expectativa de crecimiento soporta.

7.3.2 AUTOMIZACIÓN DE LAS PRUEBAS FUNCIONALES

El objetivo de las pruebas de automatización consiste en el uso de software especial (casi siempre separado del software que se prueba) para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados. La automatización de pruebas permite incluir pruebas repetitivas y necesarias dentro de un proceso formal de pruebas ya existente, es validar cuando el comportamiento observado del software que se probó a nivel de producción cumpla o no con sus especificaciones. Estas pruebas son realizadas a nivel usuario, es decir, cuando el sistema ya está en la última versión de producción.

A medida que aumenta la productividad en el desarrollo del software para la entrega del sistema final, se van disminuyendo los tiempos para las pruebas funcionales del sistema, esto provoca que crezca la incertidumbre por la calidad final del producto.

La automatización de las pruebas funcionales es una alternativa interesante para las empresas que quieren asegurar cierto nivel de calidad antes de cada liberación de productos o versiones. Aportan tranquilidad al ajustar y mejorar las principales funcionalidades, ya que brindan información sobre el impacto de los cambios realizados.



xxii) Fig. 7.2: Estructura de pruebas funcionales automatizadas

7.4 PRUEBAS DE RENDIMIENTO DE SOFTWARE

Son las pruebas que se realizan, desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. También puede servir para validar y verificar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos.

7.5 PRUEBAS DE ESTRÉS

Estas pruebas son de suma importancia, ya que se debe contemplar que un sistema debe soportar una gran cantidad de usuarios que ingresan al sistema de forma continua.

Las pruebas de estrés ayudan a encontrar fallas en el sistema que simplemente no se mostrarían en una sola instancia. Las pruebas de estrés aumentan la probabilidad de encontrar un error crítico de concurrencia o un problema de la memoria.

Esta prueba se utiliza para romper la aplicación. Consiste en que una pequeña cantidad de usuarios utiliza la aplicación y después poco a poco se empieza a doblar el número de usuarios que se agregan a la aplicación y se ejecuta una prueba de carga hasta que se rompe o manifieste un comportamiento diferente al comportamiento normal.

Este tipo de prueba se realiza para determinar la solidez de la aplicación en los momentos de carga extrema y ayuda a los administradores para determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada.

7.6 PRUEBAS DE ESTABILIDAD (*SOAK TESTING*)

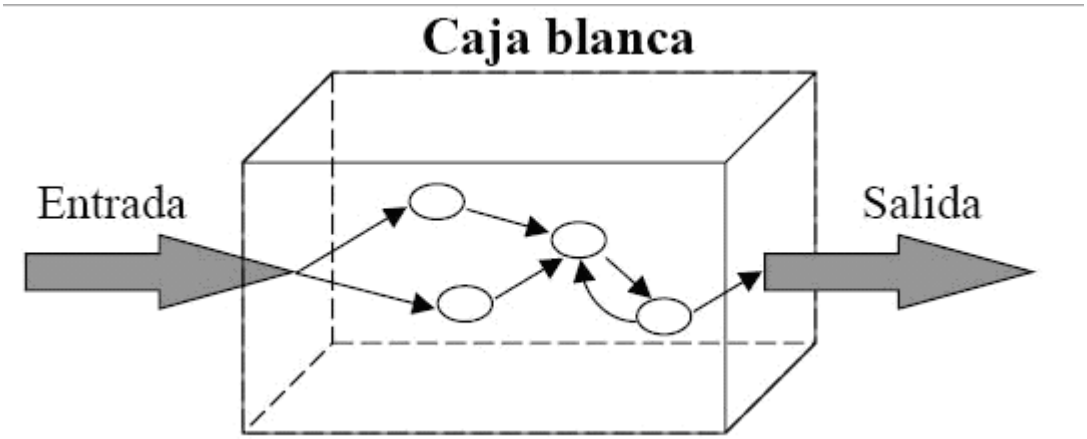
Esta prueba normalmente se hace para determinar si la aplicación o el sistema pueden soportar una carga esperada de datos continuos. Generalmente esta prueba se realiza para determinar si hay alguna fuga de memoria en la aplicación.

7.7 PRUEBAS DE PICOS (*SPIKE TESTING*)

La prueba de picos, como el nombre sugiere, trata de observar el comportamiento del sistema variando el número de usuarios, tanto cuando bajan, como cuando tiene cambios drásticos en su carga. Esta prueba se recomienda que sea realizada con un software automatizado que permita realizar cambios en el número de usuarios mientras que los administradores llevan un registro de los valores a ser monitorizados.

7.8 PRUEBAS DE CAJA BLANCA

Las pruebas de caja blanca también conocidas como pruebas estructurales o pruebas de caja transparente, se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. El *tester* escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa o sistema y cerciorarse de que se devuelven los valores de salida esperados.



xxi) Fig. 7.3: Diagrama para pruebas de caja blanca

Una ventaja es que, al realizar las pruebas, al encontrar una falla ésta se puede asociar fácilmente al módulo que no está funcionando de la forma correcta. Por lo cual simplifica el trabajo al corregir o modificar un error, sin embargo, el *tester* no puede realizar la modificación interna del sistema.

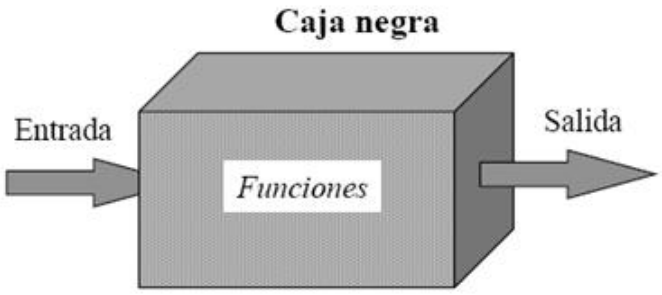
Estas pruebas se llevaban a cabo con el apoyo del equipo de desarrollo, ya que cuando el equipo de pruebas ejecuta los casos de pruebas, el equipo de desarrollo observaba el comportamiento del sistema y al terminar de ejecutar los casos de prueba que están orientados a algún modulo del sistema, el equipo de desarrollo visualiza el funcionamiento interno del sistema, para verificar que el módulo realice bien su funcionamiento y realice sus respectivos cambios en caso de ser necesario.

También brinda la ventaja que todas las partes del sistema resultan visibles, por lo cual tanto el equipo de desarrollo como el equipo de pruebas tienen conocimiento del funcionamiento interno del sistema y de sus respectivos módulos. Además, que el *tester* tiene acceso a todo la estructura o documentación del diseño.

7.9 PRUEBAS DE CAJA NEGRA

Se denomina *caja negra* a aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. Una *caja negra* nos interesará su forma de interactuar con el medio que le rodea, sin dar importancia a cómo lo hace.

En una *caja negra* debe estar muy bien definidas sus entradas y salidas, es decir, no se precisa definir ni conocer los detalles internos de su funcionamiento, en este caso el *tester* no le interesa el lenguaje de programación o la plataforma en la que se elabora el sistema.



xxii) Fig. 7.4: Diagrama para pruebas de caja negra

Una de las principales ventajas en las pruebas de caja negra es que los casos de prueba son completamente independientes de la implementación interna del sistema.

De manera que, si este llega a sufrir cualquier tipo de cambio en su estructura, las pruebas deberían proceder de forma normal sin necesidad de modificarse, ya que el cambio interno del software no debe de afectar los resultados que arroja el sistema, ni el diseño de los casos de prueba orientados a las respuestas de salida.

Las pruebas en caja negra se realizan a nivel interfaz, y solo nos importan los datos de entrada, para obtener los datos de salida esperados.

Otra ventaja es que, al finalizar las pruebas, el área de desarrollo es el que modifica y corrige errores en el sistema, con base a los resultados obtenidos, por lo cual el área de pruebas no necesita tener una orientación de programador sin embargo sí debe tener alguna noción del lenguaje o el sistema en el que desarrolla.

En estas pruebas el tester solo se enfoca a las reglas de negocio y en los resultados esperados.

7.10 PRUEBA DE CAJA GRIS

Las pruebas de caja gris también son conocidas como pruebas translúcidas, estas son una combinación de pruebas de caja blanca y pruebas de caja negra, el objetivo de estas pruebas es buscar los defectos, los cuales pueden ser debido a una estructura incorrecta, el uso inadecuado de aplicaciones o algún defecto en el código.

Un *tester* de caja gris sabe parcialmente la estructura interna y el código del sistema en desarrollo, además tiene acceso a la documentación de las estructuras de datos internas del sistema, y acceso a los algoritmos utilizados.

Las pruebas de caja gris son beneficiosas, ya que el *tester* puede modificar código en caso de ser necesario, y puede hacer la función de programador y *tester*. Lo cual beneficia en la mejora de tiempos, ya que se ahorra el proceso de esperar al equipo de desarrollo para que este corrija algún defecto y después el *tester* tenga que volver a realizar las pruebas al sistema.

7.11 PRE- REQUSITOS PARA CARGAS DE PRUEBAS

Un desarrollo establece que, de la aplicación o el sistema debe ser instalado en un entorno lo más parecido al de producción.

El entorno de pruebas de rendimiento no debe cruzarse con pruebas de aceptación de usuarios ni con el entorno de desarrollo. Esto es tan peligroso que si las pruebas de aceptación de usuarios, o las pruebas de integración o cualquier otra prueba se ejecutan en el mismo entorno, entonces los resultados no son fiables. Como buena práctica, siempre es aconsejable disponer de un entorno de pruebas de rendimiento lo más parecido como se pueda al entorno de producción.

Tampoco se deben ejecutar todas las pruebas al mismo tiempo.

7.12 PRUEBAS DE CONFORMIDAD

El servicio de *testing* de conformidad permite determinar si una aplicación es compatible con cierto estándar o estándares, puede interactuar con otras aplicaciones y cumple las normas de interfaz definidas para interactuar con el sistema deseado.
De acuerdo a los estándares identificados se elabora un plan de pruebas y se establecen los niveles de conformidad deseados.

Siguiendo las definiciones de dichos protocolos o estándares, se establecen implementaciones de referencia que son utilizadas para ejecutar los casos de prueba determinados.

Las etapas del *testing* de conformidad son:

- Planificación del proyecto
- Análisis de estándares
- Definición de pruebas y criterios de conformidad
- Búsqueda, implementación y adaptación de pruebas
- Ejecución de pruebas y reportes
- Evaluación del proyecto

Una vez finalizado el servicio, la empresa recibirá un informe final sobre las pruebas realizadas, los resultados obtenidos y los niveles de conformidad alcanzados.

7.13 TIPOS DE CASOS DE PRUEBA

La infraestructura del sistema de pruebas está conformada por cuatro niveles de *testing* que sirven como validaciones para la detección de comportamientos irregulares.

7.13.1 CASOS DE PRUEBA ORIENTADOS A TIME REPORT

Los casos de prueba estaban orientados a cargar en Time report información de las horas incurridas por recursos (externos e internos) en relación a los diferentes tipos de proyecto (proyectos cerrado, PSP-TSP, Adaevo y propio) en los que el recurso estuvo involucrado. Y de este punto se partió para diseñar casos de prueba con diferentes perfiles y con diferentes proyectos con sus determinadas validaciones.

La estructura de los casos de prueba está conformada por:

1.- Código de caso de prueba ID para enumerar los casos de pruebas

2.- Condición y datos de entrada. - Aquí vienen pre-condiciones o acciones que ya se debieron haber ejecutado o realizado, por lo cual no es necesario agregarlas en los pasos del caso de prueba.

3.- Pasos del caso de prueba. -Es una forma organizada y ordenada de detallar la acción que se está probando para cada caso de prueba, de este modo el ejecutor del área de pruebas puede llevar un mejor registro a detalle de lo que se prueba en cada paso de un caso de prueba.

4.- Resultados esperados. - Son los resultados que se desean o deben arrojar el sistema, en este punto se debe considerar que los resultados esperados se basan en la respuesta que debe arrojar el sistema a una acción controlada y el resultado debe ser el deseado, un resultado diferente implica un error.

Categoría del caso:

1.- *Happy Path* Es un campo calculado por la estimadora, indica que siempre hay que probar el llamado "escenario ideal" del sistema donde se considera que la acción que realiza el sistema debe ser la exitosa.

2.- *Escenario Alterno* Categoría en la que se evalúan los escenarios alternativos diferentes que modifican el *Happy Path* ya sea adicionando u omitiendo pasos pero que nos llevan a una operación válida, pero no siguen el curso del *Happy Path*.

3.- *Escenario de Excepción* Categoría en la que se evaluarán los diversos escenarios que nos lleven a errores u operaciones inválidas a causa de que el actor tome derivaciones del curso básico de la operación.

4.- *Reglas de Negocio* Categoría del caso en la que se verificará el cumplimiento de las reglas de negocio.

De acuerdo a los resultados indicados en la Parte de Casos de Prueba, se calculará el total de los resultados obtenidos al ejecutar los casos de prueba identificándolos con sus respectivos resultados:

Nº de casos de prueba con resultado 'Satisfactorio' y porcentaje sobre el total.

Nº de casos de prueba con resultado 'Insatisfactorio' y porcentaje sobre el total.

Nº de casos de prueba 'En Espera' y porcentaje sobre el total.

Nº de casos de prueba pendiente de concluir su ejecución, pero 'En Curso' y porcentaje sobre el total.
Nº de casos de prueba fallidos como 'Casos Erróneos' y porcentaje sobre el total.

7.13.2 NIVELES DE PRUEBAS ORIENTADOS A LA INFRAESTRUCTURA DE PRUEBAS DE TIME REPORT

El ambiente de pruebas se compone de un entorno de trabajo simulado, el cual está integrado por el sistema funcional Time report (con su última actualización), una base de datos en Oracle con datos reales y funcionales del cliente (la base de datos para el área de prueba era independiente a la de desarrollo con el fin de evitar errores que puedan ser ocasionados por un mala manipulación de la base de datos del área no correspondiente o evitar errores de modificaciones en los datos para evitar inconsistencia en las pruebas o resultados no esperados), una red local, un servidor FTP, y datos o información real de los diferentes tipos de usuarios y roles para el desarrollo de casos de prueba.

Debido a la complejidad del sistema se debía tener un área de trabajo en donde se pueda lograr un entorno similar al del área de trabajo del cliente.

Un gran paradigma y una de las principales causas por la que los tiempos de entrega y pruebas se ven afectados es por que hay un distanciamiento entre lo que el cliente quiere y lo que él necesita, y esto afecta al desarrollo del proyecto ya que a pesar de que se definieron los requerimientos y estos una vez aclarados y firmados con el cliente no deberían cambiar, esto no pasa así, siendo que había casos donde el cliente solicitaba desde el cambio en algunos pequeños detalles en algún módulo hasta incluso rediseñar la funcionalidad de este, e incluso casos donde una vez realizados los cambios a un módulo el cliente solicitaba rediseñar el módulo a como se definieron a principios del proyecto. Al iniciar el proyecto el área de pruebas y el área de desarrollo empezaron como dos áreas independientes, cada área contaba con su propio espacio en la empresa, pero esto empezó a

ocasionar problemas de comunicación entre las dos áreas lo que conllevó a un retraso en los tiempos de entrega.

La comunicación es un hecho social importante ya que surge a partir de un proceso complejo, sin duda representa una de las actividades fundamentales en el desarrollo profesional, una solución que tomó el líder de proyecto a unos meses antes de la fecha de entrega y pruebas con el cliente, fue unificar el área de pruebas y el área de desarrollo volviéndose solo un área, esta solución rompió el paradigma que existía entre la comunicación con las dos diferentes áreas, se mejoró de forma favorable los errores en códigos y se mejoraron los tiempos.

7.14 TERMINOLOGÍA BÁSICA EN SOFTWARE DE PRUEBAS

Alpha *Testing*: También llamadas pruebas alfa, son pruebas que hacen los programadores y diseñadores internamente, para comprobar que lo que están haciendo funciona bien y corregir errores.

Beta *Testing*: Las también llamadas pruebas beta son pruebas en las que el programa es ejecutado por uno o varios clientes/usuarios/voluntarios, básicamente cualquiera que no haya intervenido en el proceso de programación, cuyos ejecutables están pendientes de terminar su fase de desarrollo, o alcanzar un alto nivel de funcionamiento, pero que aún no son completamente estables.

Calidad: Se refiere al conjunto de propiedades inherentes a un objeto que le confieren capacidad para satisfacer necesidades implícitas o explícitas. (Es una fijación mental del consumidor o cliente que asume conformidad con dicho producto o servicio y la capacidad del mismo para satisfacer sus necesidades).

Incidencias: Es el número de errores que se presentan en el sistema en un terminado periodo de pruebas, (cuando el sistema no está arrojando los resultados deseados).

Error: Acción humana que produce un resultado indeseado en el programa o sistema. Es una equivocación cometida por el desarrollador.

7.15 TIPOS DE ERRORES

Defecto: Un defecto es uno o más errores que se encuentra en el programa, los cuales se manifiestan debido a la incompatibilidad que puede existir entre la versión del programa, el sistema o la plataforma de desarrollo, estos defectos también pueden ser ocasionados por un error en la sintaxis o lógica en el código.

Falla: Una falla es la discrepancia visible que se produce al ejecutar un programa con un defecto, el cual es incapaz de funcionar correctamente (no sigue su curso normal). Es una manifestación física o funcional de un error o defecto. Por lo general el mismo sistema o programa arroja una alerta de la falla. En este caso el sistema deja de funcionar.

Bug: Es un error en un programa de computadora o sistema de software que se desencadena en una determinada acción dando un resultado no deseado, a diferencia de una falla, el sistema no arroja una alerta y en algunos casos el sistema o programa puede seguir funcionando, pero de forma incorrecta.

Glitch: En el ámbito de la informática o los videojuegos es un error que, al no afectar negativamente al rendimiento, jugabilidad o estabilidad del programa, no puede considerarse un *bug*, sino más bien una "característica no prevista".

RESULTADOS

Las actividades laborales de un *tester* pueden ser muy diversas, abarcan una gran cantidad de campos dentro de la misma área. La clasificación de las labores que desarrollé en el proyecto se define de la siguiente forma:

- Realización, actualización, diseño y modificación de los casos de prueba.
- Realización, actualización, diseño y modificación del plan de pruebas
- Realización, actualización, diseño y modificación del registro de observaciones
- Ejecución de los casos de prueba.
- Ejecución de pruebas regresivas y pruebas de humo.
- Registro de resultados de los casos de prueba
- Registro, levantamiento y captura de incidencias
- Captura de videos e imágenes como evidencia en la ejecución de casos de prueba.
- Verificación de las *reglas de negocio* y los *requerimientos* aplicados al sistema, y a los casos de pruebas.
- Validación de las correctas modificaciones de las pruebas realizadas en base de datos con personal de base de datos.
- Realización de pruebas de estrés con el equipo de pruebas y el apoyo del equipo de desarrollo.
- Pre-requisitos para las pruebas de carga, desarrollo estable de la aplicación instalado en un entorno lo más parecido al de producción.

Para todas las compañías e instituciones es necesario contar con personal que pueda garantizar la calidad del producto o servicio desarrollado por alguna empresa.

Crear, analizar y realizar las pruebas del sistema con el objetivo de garantizar la calidad y brindar una solución eficaz al sistema para que este soporte una gran cantidad de usuarios y además se pueda garantizar la integridad de los datos y la información contenida en este, fue uno de los retos más importantes para el equipo.

Time report al ser un sistema robusto presenta una parte fundamental para el cliente ya que el sistema se asocia a la administración y pagos para su personal. Los diferentes tipos de pruebas, documentaciones y herramientas que realicé y aporté para conformar Time report ayudaron a mejorar el sistema y disminuir los errores, además de aumentar la calidad del producto final antes de entrar a producción.
El acercamiento al área de desarrollo jugó un papel fundamental para la creación de algunos casos de prueba, también para la solución e identificación de fallos, el trabajar de forma organizada con el equipo de desarrollo también ayudó a tener un mejor entendimiento de los diferentes problemas que se presentaron y mejorar los tiempos.

También sirvió de gran ayuda contar con el equipo de base de datos, quienes nos apoyaron para validar los datos que usábamos en las tablas de pruebas, para que estos fueran íntegros, y no se modificaran o tuvieran un comportamiento no deseado.

Cuando el producto entró a producción con el cliente los errores que se manifestaron fueron mínimos, y aunque el equipo de pruebas junto con el equipo de desarrollo colaboraron con

el cliente para corregir estos errores de forma inmediata, siempre se debe contemplar que ningún sistema puede ser perfecto e incluso cuando un conjunto de personas realiza gran cantidad de pruebas, siempre van a haber factores o eventos que puedan generar algún error, falla o resultado no esperado, al final se cumplió el objetivo de disminuir los errores del sistema y entregar un producto con la mejor calidad posible.

CONCLUSIONES

Un sistema puede llegar a ser muy simple o muy complejo, sin embargo, no será un sistema perfecto, no al menos en las primeras etapas de desarrollo, y posiblemente cuando el sistema esté en producción este no va a ser perfecto. Por eso, el trabajo del *tester* es disminuir o erradicar todas estas fallas antes de que el sistema salga a producción, esto con el fin de garantizar calidad, lo más importante para una empresa es su imagen y la calidad de su producto. Un producto ineficiente o imperfecto implica pérdidas, y la empresa pierde credibilidad en sus productos.

El reto principal para un *tester* es contemplar la cantidad de escenarios posibles donde el sistema presente un comportamiento diferente al deseado, además un *tester* debe contar con un perfil orientado a la programación pero también parte de su formación va dirigida a un perfil con habilidades sociales, enfocado a desarrollar su imaginación ya que por más ridículo que esto suene un *tester* debe tener o contemplar el mayor número de escenarios factibles por lo cual un sistema podría no funcionar, no en todos los casos el *tester* debe ser programador ya que puede ir orientado más al aspecto financiero y las reglas de negocio.

Actualmente no se puede decir que exista una forma eficiente para garantizar el correcto comportamiento del sistema y que este sea libre de errores, pero se pueden tomar medidas preventivas y también saber proponer soluciones.

Es por esto que los *testers* juegan un papel importante y este rol ha ido evolucionado de forma significativa en los últimos años, sobre todo para el desarrollo del software, esto por la alta competitividad que existe entre empresas, no pueden darse el lujo de cometer algún error por más simple que este pueda ser.

El participar como *tester* en el área de pruebas para el proyecto Time report fue todo un desafío a nivel profesional, ya que el aprender de una área relativamente nueva para mí me llenó de incertidumbres, no sabía que esperar, si lo que debía hacer o estaba haciendo era lo correcto o si podía hacerlo de otra forma, al ir ganando experiencia desarrollé mayor participación para las actividades del área de pruebas, incluso brindar aportaciones asertivas sobre el diseño y la implementación de los casos de pruebas, así como considerar cuales pruebas no eran necesarias o se podían mejorar.

A pesar de que todo el equipo de pruebas tuvo una buena organización en el proyecto y tener bien medidos los tiempos de las actividades, en algunas ocasiones no se cumplieron las fechas para el periodo de pruebas, debido a que estas pruebas se extendían uno o dos días, además de que también llegaban a existir atrasos por parte del equipo de desarrollo, esto afectaba cuando al equipo de pruebas por que se prolongaba la liberación del sistema para poder realizar los casos de pruebas y en ocasiones cuando se estaba por finalizar las pruebas del sistema o el equipo de desarrollo estaba por liberarlo, el cliente solicitaba cambio o nuevas modificaciones, lo cual atrasaba los tiempos tanto del equipo de desarrollo

como del equipo de pruebas, pero el tiempo para la entrega del proyecto con el cliente no se extendió a menos que realmente fuera necesario por los cambios solicitados.

Al final el producto que se entregó cumplió con los requerimientos y las necesidades del cliente, brindando calidad al sistema.

REFERENCIAS

[1] Myers, Glenford J., (2004). *The art of software testing*. [en línea]. New Jersey: John Wiley & Sons, Inc., Hoboken. Disponible en: http://barbie.uta.edu/~mehra/Book1_The%20Art%20of%20Software%20Testing.pdf [23/11/2015] Brinda un enfoque de buenas metodologías para realizar pruebas.

[2] Goucher, Adam. (2010). *Beautiful Testing: Leading Professionals Reveal How They Improve Software*. Primera edicion [versión digital]. Gravenstein Highway North, Sebastopol: O´Reily Media, Inc. [23/11/2015]

[3] Jiantao Pan. (1999). *Software Testing.* [en línea]. Pittsburgh (Pensilvania): Carnegie Mellon University: Disponible en: http://users.ece.cmu.edu/~koopman/des_s99/sw_testing/ [23/11/2015] Reseña de diferentes prácticas y conceptos para mejorar las técnicas de testeo.

[4] Kaner Cem. (2006). *Exploratory testing.* [en línea]. Stanford, California: Victor R. Basili, University of Maryland: Disponible en: http://www.kaner.com/pdfs/ETatQAI.pdf [23/11/2015] Diversas técnicas para realizar diferentes tipos de pruebas.

[5] Norman Parrington and Marc Roper. (1989) *Understanding Software Testing* [version digital]. University of Michigan: John Willey & Sons.

[6] Filippos I. Vokolos, and Elaine J. Weyuker. (2000) *Performance testing of software systems*. [en línea] IEEE Press Piscataway, NJ, USA: Disponible en: http://es.slideshare.net/Softwarecentral/experience-with-performance-testing-of-software-systems Metodología a base de experiencia para la aplicación del testing en sistemas

[7] Boris Beizer. (1990). *Proceedings of the first international workshop on Software and performance*. [version digital] Software Testing Techniques. Second edition.

[8] Harry Koehnemann, and Timothy Lindquist. (1993) *Towards target-level testing and debugging tools for embedded software* [versión digital], Arizona State University

[9] R. Edward Freeman. (1984) *Strategic Management: A Stakeholder Approach*, http://www.researchgate.net/publication/228320877_A_Stakeholder_Approach_to_Strategic_Management [23/11/2015] Brinda un enfoque de direccionamiento estratégico del negocio para los stakeholders.

GLOSARIO DE TÉRMINOS

A

Actividad cognitiva. - Es un proceso a través del cual el sujeto capta los aspectos de la realidad, a través de los órganos sensoriales con el propósito de comprender la realidad. Involucra armónicamente a todas las funciones mentales, percepción, memoria, pensamiento, lenguaje, creatividad, imaginación, intuición, interés, atención, motivación, conciencia e incluso creencias, valores y emociones

Adaevo. - Proyecto particular del cliente, el cual no se consideró relevante conocer para la elaboración del proyecto y por tal motivo el cliente no reveló información.

AMS. - (Servicio de mantenimiento de aplicaciones) representa un enfoque avanzado de servicios TI mediante el cual asume la responsabilidad a medio/largo plazo del conjunto de tareas y actividades relativas tanto al desarrollo y mantenimiento de aplicaciones como al soporte y evolución de las mismas

C

Concurrencia. - Es la convergencia de datos o información que ocurre cuando se ejecutan múltiples tareas o conjunto de procesos creados por un único programa.

Control de calidad. - El control de calidad son todos los mecanismos, acciones y herramientas para detectar y corregir la presencia de los errores encontrados.

Core-team. **-** Se puede definir como un grupo de personas esenciales que van a ser los responsables máximos de las decisiones dentro de un proyecto y desempeñar una labor o función en este.

D

Discrepancia. - produce una acción o comportamiento diferente al que debería de tener.

E

Escenarios alternos. - Nos definen las diferentes posibilidades que podrían ocurrir cuando estos escenarios no estén relacionados a las reglas de negocios, pero pueden presentarse o afectar de forma negativa al sistema.

Escenarios funcionales. - Donde se prueban los requisitos funcionales definido para el sistema.

E200.- Nomenclatura utilizada para identificar el documento BUSINESS CASE el cual nos

detalla el alcance del proyecto del cliente, cabe mencionar que los diferentes documentos de TOWA tienen una nomenclatura que sirve como un identificador del documento mismo.

F

FLEP-PT.- Metodología para las buenas prácticas del proceso del desarrollo del software.

Flujos de ejecución. - Hace referencia a cada línea de código que será ejecutada por el procesador.

Framework. - En el desarrollo, un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, lenguaje intérprete, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

FTP. - (File transfer protocol), protocolo de red para el intercambio de archivos entre sistemas conectados por red.

G

Gestión de proyecto. - Consiste en la disciplina del planeamiento, la organización, la motivación, y el control de los recursos con el propósito de alcanzar uno o varios objetivos. El primer desafío para la gestión de proyectos es alcanzar la meta del proyecto y los objetivos dentro de las limitantes conocidas. Las limitantes o restricciones primarias son el alcance, el tiempo, la calidad y el presupuesto.

H

Happy path. - Escenario donde se considera que todo lo que se haga en el sistema a desarrollar esta bien o es correcto

HTML. - (Lenguaje de marcas de hipertexto), es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML)

I

IDE. - (Integrated development environment**)** es una aplicación o sistema informático que proporciona servicios integrales para facilitar al desarrollador o programador el desarrollo de software.

Incidencia. - Suceso o circunstancia que ocurre en el desarrollo del sistema que puede afectar o influir en alterar su comportamiento.

Instalación. -El momento donde el sistema se instala por primera vez con el cliente, en este caso el sistema todavía está en un periodo de prueba antes de entrar a producción.

Instancia. - Una instancia se refiere a una realización específica de una clase determinada en la programación orientada a objetos, por lo general se ocupa el término cuando se crea un objeto a partir de una clase.

Investigaciones empíricas. - La palabra empírica significa información obtenida por medio de la experiencia, observación de los experimentos. El tema central en el método científico es que todo aporte debe ser empírico, lo que significa que es basado en la evidencia. Teniendo esto en cuenta, la palabra "empírica" también se refiere a trabajar con hipótesis que pueden comprobarse mediante la observación y los experimentos.

Iteración. - Repetir un proceso con el objetivo de alcanzar una meta deseada, objetivo o resultado.

J

JavaScript. - JavaScript (abreviado comúnmente "JS") es un lenguaje de programación del estándar ECMAScript.
Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámica, utilizado en la programación orientado a objetos, basado en prototipos, imperativo y dinámico.

L

LAN. - (local area network) Es un grupo de equipos que pertenecen a la misma organización y están conectados dentro de un área geográfica pequeña a través de una red, generalmente con la misma tecnología (la más utilizada es Ethernet).

Layouts. - Suele utilizarse para nombrar al esquema de distribución de los elementos dentro un diseño, es decir, son los elementos que aparecerán en una pantalla

LOC.- Son el total de líneas de código de un determinado sistema.

P

PEP. - Es una cantidad monetaria destinada a un determinado proyecto.

Periodo de pruebas. - Hace referencia al periodo estimado en el que se deberían estar realizando las pruebas, estas fechas normalmente son determinamos ya sea por el cliente y el líder de proyecto, también se involucran varios factores como el presupuesto del proyecto, la fecha para desarrollar el proyecto, y las personas que participan en éste.

Pipelining.- La segmentación, es un método por el cual se consigue aumentar el rendimiento de algunos sistemas electrónicos digitales, La segmentación consiste en descomponer la ejecución de cada instrucción en varias etapas para poder empezar a procesar una instrucción diferente en cada una de ellas y trabajar con varias a la vez.

Producción. - Hace referencia cuando el sistema desarrollado se encuentra en sus primeras etapas de funcionamiento para el cliente.

Pragmatismo. - Hace referencia a lo práctico, la ejecución o la realización de las acciones, no a la teoría o la especulación. El pragmatismo se caracteriza por la insistencia en las consecuencias como manera de caracterizar la verdad o significado de las cosas, Rechaza la existencia de verdades absolutas, las ideas son provisionales y están sujetas al cambio, a la luz de la investigación futura.

Proyecto cerrado. - Consiste en que el cliente contrata a una o varias empresas para desarrollar un proyecto para su empresa. Donde un ejecutivo del cliente con rango alto se encarga de llevar el proyecto y darle seguimiento.

R

Reglas de negocio. - Describen las políticas, normas, operaciones, definiciones y restricciones presentes en una organización y que son de vital importancia para alcanzar los objetivos misionales.

Requerimientos. - Término utilizado para hacer referencia a las necesidades solicitadas por el cliente para implementarlas en el sistema. Aquí se pueden involucrar necesidades técnicas y aspectos administrativos orientados a las reglas de negocio.

Responsable. - Persona de confianza que podrá realizar las actividades que le corresponden al jefe inmediato.

S

SIAPRO. - Sistema utilizado por la empresa para dar de alta, baja, bajas definitivas y cambios del personal de la empresa, este sistema tiene independencia con Time, eso quiere decir que si un jefe inmediato da de baja a su recurso de un proyecto este recurso solo se dio de baja en Time, pero no en SIAPRO.

Software. - Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

SPG. - Sistema para la gestión y control de peps, donde se cargan peps y órdenes de compra del cliente en formato excel.

Stakeholder. - Termino para referirse a quienes pueden afectar o son afectados por las actividades de una empresa, como pueden ser los trabajadores de esa organización, sus accionistas, los sindicatos, las organizaciones civiles y gubernamentales que se encuentren

vinculadas, etc.

T

TCO. -Método para calcular y estimar los costos directos o indirectos totales relacionados con la compra de programas o equipos informáticos

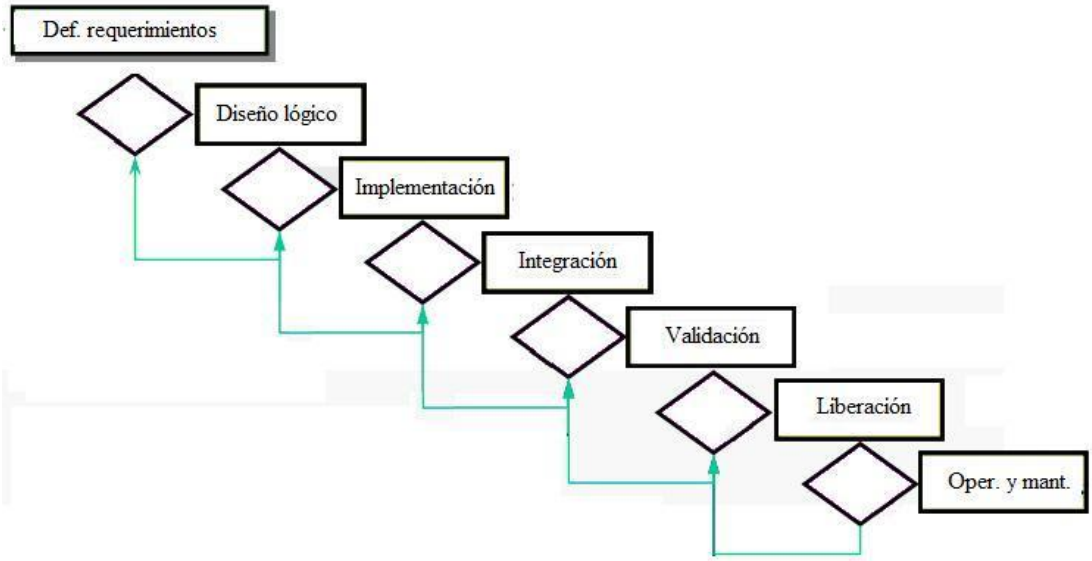
Tester. - Es la persona que realiza las funciones de planificación para las pruebas de software con la finalidad de identificar, detectan y solucionar errores, además de evaluar el funcionamiento del software.

X

XML. - (extensible markup language) nos permite definir la gramática de lenguajes específicos para estructurar documentos grandes y almacenar datos en forma legible**.**

ANEXO A

MODELO EN CASCADA PARA LA CONFIABILIDAD Y DESARROLLO DEL SOFTWARE



xxiii) Fig. A.1: Modelo en cascada para el desarrollo de software

El modelo en cascada consta de las siguientes fases:

1. Definición de los requerimientos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle.
2. Diseño de software (diseño lógico): Se hacen particiones del sistema, en sistemas de software o hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de los componentes del sistema.
3. Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.
4. Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se entrega el conjunto probado al cliente.
5. Validación de pruebas: Se validan las pruebas que han sido realizadas, eso con el fin de asegurar que lo que ya funciona, siga funcionando, esto llega a ocurrir en caso de que se tenga que modificar alguna funcionalidad del sistema. (generalmente esta

fase viene implícitamente en la parte de integración y pruebas de sistemas).

6. Liberación: El sistema ya se encuentra instalado de forma productiva con el cliente. Donde en teoría el sistema debería ser funcional y libre de errores. (generalmente esta fase viene implícitamente en la parte de integración y pruebas de sistemas debido a que ya se considera que el sistema no debería de tener fallas, aquí se involucra la instalación en sistema con el cliente).

7. Operación y mantenimiento: El sistema es puesto en marcha, se realiza mantenimiento y correcciones en caso de encontrar alguna falla o error que no se contempló en el para las pruebas o de una posible falla.

La interacción entre fases puede observarse en la Fig. 12.1 cada fase debe de estar aprobados y verificados antes de presentarse el trabajo con el cliente.

Una fase no comienza hasta que termine la fase anterior y generalmente se incluye la corrección de los problemas encontrados en fases previas.

Las características principales de este paradigma son:

Los procesos específicos y ejecutables que constituye el primer prototipo del sistema. Posteriormente, conforme se va avanzando se convierten en las implementaciones del sistema, en el último paso se obtiene una implementación en un lenguaje de programación determinado. El mantenimiento se realiza sobre la especificación (no sobre el código fuente)

Algunas observaciones sobre el desarrollo formal del sistema:

- Permite demostrar la corrección del sistema durante el proceso de transformación. Así, las pruebas que verifican la correspondencia con la especificación no son necesarias.
- Es atractivo sobre todo para sistemas donde hay requisitos de seguridad y confiabilidad importantes.

ALTA DISPONIBILIDAD

Alta disponibilidad es considerada un protocolo de diseño del sistema, su implementación es asociada para asegurar un grado absoluto de continuidad operacional durante un período de medición dado (el sistema debe poder ser utilizable o accesible la mayor parte del tiempo necesario por los usuarios y este siempre debe estar funcionando, alguno de los factores básicos es que se debe contemplar un sistema robusto y considerar la cantidad de personas que pueden acceder a éste). Y con disponibilidad se hace referencia a la habilidad de la comunidad de usuarios para acceder al sistema, someter nuevos trabajos, actualizar o alterar trabajos existentes o recoger los resultados de trabajos previos. Si un usuario no puede acceder al sistema se dice que está no disponible. El término tiempo de inactividad (*Down time*) es usado para definir cuándo el sistema no está disponible.

TOLERANCIA A FALLOS

El concepto de tolerancia a fallos (*failover*) hace referencia a la capacidad de un sistema de acceder a la información, aun en caso de producirse algún fallo o anomalía en el sistema. Le permite a un sistema seguir funcionando correctamente en caso de fallo de uno o varios de sus componentes. Si disminuye su calidad de funcionamiento, la disminución es proporcional a la gravedad de la avería.

Particularmente se busca una alta disponibilidad. Aquí pueden entrar algunos conceptos como planes de emergencia referentes al respaldo (*back up*) además de contar con un fácil acceso a la información con medios alternos.

TOLERANCIA A DESASTRES

El concepto de tolerancia a desastres nace de las necesidades de las empresas al tener un respaldo de la información del entorno de trabajo de la empresa, en caso de que pueda ocurrir una catástrofe o un desastre que impida mantener actividades en la empresa, la tolerancia a desastres se puede dividir en los dos siguientes conceptos:

Hot site: Es un servicio comercial de recuperación de datos frente a desastres que permite a las empresas mantener sus operaciones informáticas y de red en caso de un fallo de los equipos o de las instalaciones, por lo general es un sitio remoto y seguro que guarda los datos duplicándolos como respaldo, estos datos se copian a la par de los cambios que se realizan en el sistema original, la ventaja de un hot site es que este sistema es funcional, es decir que en el momento que falla el sistema primario, el hot site empieza a trabajar como si fuera el sistema base en cuestión de minutos.

Cold site: Es un servicio similar al hot site, pero con la diferencia de que es el cliente quien suministra e instala el equipamiento necesario para poder reactivar las operaciones. Este sitio frío es más barato, pero también necesita más tiempo para poder ponerse en marcha y recuperar la plena operatividad después del desastre, además estos respaldos no siempre contienen la última versión o actualización del sistema, es decir que si se realizó un respaldo hace un mes y el sistema falla. Al instalar el cold site, este no tendrá la información referente a un mes.