UNIVERSIDAD CATÓLICA DE SANTA MARÍA

FACULTAD DE CIENCIAS E INGENIERÍAS FÍSICAS Y FORMALES

"PROPUESTA DE UN MÉTODO PARA EJECUTAR PRUEBAS DE VALIDACIÓN DE SOFTWARE EN EL DESARROLLO DE LOS SISTEMAS INFORMÁTICOS PARA UNA ENTIDAD FINANCIERA EN LA CIUDAD DE AREQUIPA."

Autor: Polanco Paredes Karen Dayan

Trabajo de investigación para optar el Título,
Profesional de Ing. de Sistemas

Arequipa – Perú

2014

DEDICATORIA

Dedico mi trabajo de tesis:

A Dios mi Creador y Padre, por darme la vida del día a día y permitir que llegue hasta este punto y no caer en el camino, por darme la capacidad para terminar la elaboración de este trabajo.

A la mamita de Chapi; por colocar a las personas correctas como guías para esta tesis, por darme la fuerza y valentía para poder continuar, gracias virgencita que con tu mirada de madre me daba el apoyo mediante las personas que estuvieron conmigo durante este proceso. Gracias mamita por tus bendiciones derramadas en la familia que me diste y mis amigos que están conmigo.

A mi mamá Sirley que con su paciencia y todo su amor, permitieron que culmine el trabajo al igual que mi papá Henry, por darme la educación, y los medios para salir adelante. Ellos los pilares fundamentales para mi vida quedaré siempre agradecida por las lecciones enseñadas para ser mejor persona y ser mi ejemplo de superación. Gracias por creer en mí, y ayudarme siempre y en todo momento, los amo.

A mis hermanos; Shirley y Jonathan, gracias manitos por todo ese cariño inmenso que me dan y es correspondido, a pesar de las lejanías siempre estamos cerca con todo el corazón, gracias por su preocupación y ser mi modelo a seguir en todo aspecto. No tengo mejores amigos que ustedes, los quiero mucho, gracias por no dejarme sola.

Al Ing. Fernando Paredes, por su tiempo dedicado y el empeño que se puso para sacar adelante el tema de tesis y toda su elaboración. Gracias Ing.!!

A mis compañeros de trabajo de Compartamos Financiera, por enseñarme con las experiencias y su tiempo dedicado en ayudarme cuando necesite de él.

Y a todas aquellas personas que estuvieron cerca de mí y aún están, familiares, amigos; motivándome para crecer y no rendirme. Gracias a todos aquellos que fueron piezas importantes, gracias por su paciencia. A ustedes con mucho cariño.

INDICE GENERAL

RESUMEN

El método propuesto para la realización de pruebas en una institución financiera se basa en la descripción de requerimientos y el enfoque funcional. Validar; funciones, entradas y salidas, las que corresponderán a lo solicitado en el requerimiento

Se diseña en *test case* para pruebas funcionales, al momento de decidir si el requerimiento presentado es viable, éste deberá cubrir el máximo de número de entradas, analizar los valores límite (explorar las condiciones límite de un programa y producir un mejor resultado para detectar errores) y conjeturar sobre la posibilidad de errores (lista de posibles equivocaciones del desarrollador).

En el momento, que el desarrollador, realice el análisis del requerimiento, se realizarán casos de uso, mediante diagramas de actividades para el diseño de pruebas de escenario, y así validar las funciones solicitadas con las entradas y salidas que corresponderían.

Una vez, se llegue a la etapa de ejecución, se diseñará un test case de pruebas de testeo dinámico, en donde se expandirá un histórico de pruebas al momento que se ejecute el programa a probar.

ABSTRACT

The proposed testing method in a financial institution is based on the description of requirements and functional approach. Validate; functions, inputs and outputs, which correspond to the request in the request

It is designed to test case for functional testing, when deciding whether the requirement presented is feasible, it must cover the maximum number of entries , analyze the limit values (explore the boundary conditions of a program and produce a better result for errors) and speculate on the possibility of errors (list of possible mistakes developer) .

At the time, the developer, perform requirement analysis, use cases will be made through activity diagrams for test design stage, and the functions required validating the inputs and outputs that correspond.

Once it reaches the stage of execution, a dynamic test case testing test, where a historic test at the time the program is run it will expand testing will be designed.

CAPÍTULO I

PLANTEAMIENTO TEÓRICO

1.1. Título Descriptivo del Proyecto.

"Propuesta de un método para ejecutar pruebas de validación de software en el desarrollo de los sistemas informáticos para una entidad financiera en la ciudad de Arequipa."

1.2. Descripción del Problema

Una vez generado el código fuente, el software debe ser probado para descubrir y corregir el máximo de errores posibles antes de su entrega al cliente. (Pressman, 2005). En la actualidad, el ciclo de vida del desarrollo de sistemas en la división de Tecnología de Información (TI); se entiende como etapa de pruebas, cuando el desarrollador termina de implementar lo solicitado en el requerimiento; realiza pruebas de unidad, para luego realizar pruebas con el usuario (validación) llegando finalmente a la etapa de certificación (verificación).

En la entidad financiera; no se cuenta con un método apto para realizar pruebas, el error se detecta en el usuario final y no por el personal de desarrollo, ni el personal de validación ni verificación, desprestigiando a la división de Tecnología.

La continuidad del negocio financiero es vital para la atención a los clientes, sabemos que los sistemas informáticos de entidades financieras deben ser altamente confiables y. Por lo tanto el controlar y mejorar la eficiencia y eficacia al realizar las modificaciones del software es muy importante.

Por otro lado; el desarrollo de software (incluyendo las modificaciones) tiene un reto interesante al momento de considerar los requerimientos del sector financiero,

porque las organizaciones financieras son altamente dinámicas, constantemente cambian sus procedimientos, se expanden, se fusionan, amplían sus servicios y productos, etc., esto sumado a que constantemente sus procesos internos son regulados con nuevas normas o estándares definidos por entes de control externos.

Este dinamismo se refleja en su estructura tecnológica e informática, y hace que al momento de emprender los desarrollos encontremos una serie de características recurrentes.

Una de estas características es la gran cantidad de sub-sistemas, cada área tiene sus propias aplicaciones en diversas plataformas, base de datos o tecnologías. Un creciente número de clientes y operaciones, y la complejidad del negocio hace necesario mantener y procesar gran cantidad de datos. La confidencialidad de la información y lo sensible de esta hace que los métodos de intercambio de información sean limitados y deban ser blindados, por limitaciones políticas y técnicas.

También sucede que los procesos demandan gran cantidad de trabajo manual en las plataformas, como por ejemplo los procesos de cierre, en los que el procesamiento es en lotes sobre toda una base de datos y debe ser cuidado por el personal.

Uno de los problemas más importantes que afecta al *testing* de software es que se requieren diferentes latencias de integración. La periodicidad de los procesos es muy variable: diario, semanal, quincenal, mensual, trimestral, semestral, anual lo que hace difícil la sincronización de varis subsistemas.

La presente tesis, investiga y describe la manera óptima de obtener un método dable y óptimo para realizar pruebas de validación adecuada respecto a lo solicitado.

1.3. Área Científica a la que corresponde el Problema

Ingeniería de Software

1.4. Tipo y Nivel de Investigación.

Tipo de Investigación Científica: Tecnológica

Nivel de Investigación: Experimental

1.5. Objetivos

1.5.1. Objetivo General:

Elaborar una propuesta de un método testing para poder desarrollar pruebas de validación de los sistemas informáticos para una Entidad Financiera en Arequipa.

1.5.2. Objetivos Específicos:

- Desarrollar un plan de pruebas para mejorar las aplicaciones requeridas.
- Evaluar la funcionabilidad del software a prueba de fallos de acuerdo a los requerimientos del usuario solicitante.
- Disminuir los errores en la producción de los Sistemas Informáticos Financieros.
- Dar una mayor calidad al software.

1.6. Alcances y Limitaciones

Alcance: Sistemas Informáticos en entidades financieras de la ciudad de Arequipa.

Limitaciones: Se probará la propuesta en el módulo de clientes en una empresa financiera de la ciudad de Arequipa

1.7. Justificación

De acuerdo a su naturaleza, las entidades financieras son altamente dinámicas, sobre todo en lo que corresponde a su área de Tecnologías de la Información y Comunicaciones; constantemente hay nuevos requerimientos debido a que cambian sus procedimientos, se expande la entidad, se fusionan sus servicios, se amplía su cobertura,

se modifican sus servicios y productos, etc. Esto se refleja en que permanentemente existen requerimientos para modificar los módulos y hay que enfrentarlos, modificarlos y ponerlos en producción. Además este trabajo es constantemente controlado por auditoría de acuerdo a sus normas y estándares.

Entonces se justifica el desarrollo de una propuesta para mejorar el rendimiento de las pruebas de software debido a que mejorará la calidad del mismo y el manejo de la información en la entidad financiera.

CAPÍTULO II

MARCO TEÓRICO

2.1. Estado del Arte

- ***A proposed model for software testing documentation incorporated in the software quality assessment tool derived of research (propuesta de un modelo para la documentación de pruebas de software incorporado en la herramienta de evaluación de calidad de software derivado de actividades de investigación). Nelson Enrique León Martínez, Mónica Janeth Blanco Díaz, Diego Armando Villarreal Díaz, Luis Carlos Gómez Flórez. Grupo de Investigación en Sistemas y Tecnologías de la Información. Universidad Industrial de Santander Bucaramanga – Colombia. 2008.***

A través de este "artículo" se da a conocer la propuesta de un modelo para la documentación de "testing" de software, que pretende ser una herramienta complementaria con el modelo de gestión y evaluación de la calidad de software derivado de actividades de investigación,

Este artículo analiza un modelo para documentar las pruebas de software que se desarrolla dentro de los grupos de investigación. Estas pocas veces se planifican desde el inicio de la investigación, muchos de quienes los implementan no son conscientes de la importancia y el beneficio que representa para los usuarios finales las herramientas desarrolladas, ni el impacto socio-económico que podría tener. También estudian la evaluación de calidad que se deja como una actividad opcional y no vital del "testing" de software. También en lo relacionado con calidad, las pruebas

de software juegan un papel muy importante a la hora de realizar una correcta evaluación.

- ***Software testing fundamentals—concepts, roles, and terminology. (fundamentos de las pruebas de software-conceptos, roles y terminología) John E. Bentley. Corporate Data Management and Governance. Wachovia Bank. 201 S. College Street, NC-1025. Charlotte NC 28210. 2005.***

  Este artículo propone un conjunto completo de herramientas de desarrollo de aplicaciones para la construcción independiente de aplicaciones cliente-servidor y de aplicaciones habilitadas para Internet. Además se propone una metodología para la capacitación y formación en el uso del software. Además se hace un estudio sobre cierta tendencia que existe para construir aplicaciones que resultan ser armas de doble filo, debido a que no sólo se puede crear buenas aplicaciones, sino que existe la posibilidad (y tendencia) para crear aplicaciones que frustran a los usuarios, usan mal los recursos informáticos y, consecuentemente, dañan la credibilidad de los ingenieros de software. Las pruebas formales y el "testing" de software ayudan a evitar errores en la implementación del software. Para aquellos no familiarizados con el tema, este documento puede servir como primer paso para aprender acerca de un enfoque más formal y riguroso para las pruebas de software.

- ***Software testing methods and techniques. (métodos y técnicas para pruebas de software). Jovanović, Irena. IPSI Bgd Internet Research Society. New York, Frankfurt, Tokyo, Belgrade. January 2009 Volume 5 Number 1 (ISSN 1820-4503)***

  En este "artículo" se describen, en forma general, los principales métodos y técnicas de pruebas de software que se realizan para desarrollar software de calidad. Además, se muestra una clasificación general, en resumen, de los dos métodos de

prueba más comunes y, generalmente, de uso obligatorio en el “testing de software”: Las pruebas de caja negra y las pruebas de caja blanca. Se presenta, además, las técnicas y procedimientos de uso de mayor frecuencia para ambos métodos.

En el caso de las técnicas de Caja Negra se explican los términos básicos de mayor uso y que se aplican usualmente: partición equivalente, análisis de valor límite, causa-efecto, técnicas gráficas, y pruebas de comparación;

Por otro lado las técnicas de Caja Blanca tienen los siguientes términos usados con mayor asiduidad: camino de bases de pruebas, pruebas de lazo, y la estructura de prueba de control (Control Testing). Por otro lado se ilustra la clasificación de la IEEE Computer Society.

- ***Crear aplicaciones de sistemas de pruebas de control de hardware (hil). Universitaria de Investigación y Desarrollo. Congreso Académico UDI 2012. ISSN: 2027 – 3703. Fecha de Publicación: Sep 19, 2013***

  Este “artículo” se basa sobre los tutoriales que hablan sobre la elección de interfaces de E/S y sobre cómo usar inserción de fallas de hardware. En este caso el estudio versa sobre las aplicaciones de pruebas HIL (Control de Hardware) que utilizan estas interfaces para comunicar con la unidad de control electrónico (ECU) que ha sido evaluada y probada y están compuestas de dos sistemas principales de ejecución: una aplicación determinística en tiempo real y una aplicación principal.

  Este trabajo describe ambos componentes y las tareas que realizan, desde los requerimientos hasta la integración del sistema, pasando por la configuración y programación.

- ***The personal software process (el proceso personal de software). Watts Humphrey. Technical Report. CMU/SEI-2000-TR-022. Hernan Berinsky, Francisco Facioni,***

Pablo Sobrino, Verónica Sznek. Buenos Aires, 2 de junio de 2008

El "Personal Software Process" es una propuesta que nace de la necesidad de reducir los costos de "testing" y arreglo de software.

Se propone un conjunto de métodos, formularios y scripts que guían a los ingenieros en la realización de un proyecto de software. Además se concentra en el trabajo de cada ingeniero individualmente, el estudio fue diseñado para ser utilizado con cualquier lenguaje de programación y cualquier metodología de diseño.

Se propone mostrar a los ingenieros cómo manejar la calidad durante desarrollo, cómo analizar los resultados de cada trabajo y cómo utilizar esos resultados para siguientes proyectos.

Los elementos de PSP son:

- Scripts: definen pasos para cada etapa del proceso.
- Logs y Forms: plantillas para registrar y almacenar datos
- Guía Standard: indica la manera de realizar el trabajo.
- Adaptación de CMM a escalas pequeñas.

- ***Trabajo Informe. Rivera Guillén Briggi. Programa Profesional de Ingeniería de Sistemas. Universidad Católica de Santa María. 2012***

A pesar de que la empresa de seguros contaba con una metodología de pruebas de software, la calidad de los productos y requerimientos entregados por las diversas empresas proveedoras de desarrollo de software no era la adecuada, ya que, se presentaban constantes errores luego de la puesta en producción de un software. En tal sentido, la empresa requería de un servicio de consultoría para la revisión, análisis y optimización del proceso de pruebas del software.

DATCO Perú es una empresa filial de uno de los grupos más importantes de Tecnologías de la Información de la Región Andina y del Cono Sur de Latinoamérica, con más de 8 años en el rubro de las tecnologías de la información y comunicaciones. El Grupo DATCO está compuesto por: Empresas de Tecnologías de Información: DATCO Argentina, DATCO Chile, DATCO PERÚ, DATCO Uruguay, DATCO Ibérica. Empresas de Comunicaciones y Transmisión por Fibra Óptica: Velocom Argentina, Silica Networks.

DATCO Perú, trabaja para solucionar las necesidades de las empresas medianas y corporativas en los segmentos de comunicaciones, banca y finanzas, seguros, gobierno, manufactura, distribución etc. DATCO Perú provee soluciones de negocios basadas en tecnologías de la información, generando valor a sus clientes y con la expectativa de consolidarse como líder en el mercado a través de la excelencia de sus servicios, con un equipo humano altamente calificado y como socio de negocios de clase mundial.

- ***Tesis: Desarrollo de un modelo que permita la gestión cuantitativa de procesos de mantenimiento de software, Infantas Luque Tatiana Sofía. Programa Profesional de Ingeniería de Sistemas. Universidad Católica de Santa María. 2005***

  Este trabajo de tesis tiene como objetivo presentar un modelo específico para el proceso de mantenimiento de software, tanto a nivel de gestión como de implementación, el cual, en resumen, es una aplicación de diferentes estándares, modelos y metodologías existentes en el mercado mundial.

  El modelo base que se ha tomado es CMM (Modelo de Madurez de Capacidades), que nos indica que hacer, mas no la forma de hacerlo. Otros modelos se han tomado para las fases de implementación y gestión y sí nos indican la forma

de hacer las cosas; y estos son: Puntos de Función, DRA (Desarrollo Rápido de Aplicaciones), métricas específicas para el mantenimiento de software, etc.

La gestión de los requerimientos nos permitirá obtener los datos concernientes a la implementación de los mantenimientos, y la herramienta desarrollada, nos apoyará en la automatización del almacenamiento de dichos datos. De esta manera se podrán medir los resultados y optimizar el proceso descrito, o tomar acciones preventivas o correctivas en los mantenimientos en curso y los que se implementarán a futuro, teniendo un mejoramiento continuo del proceso y por ende del producto, lo cual implica un mejoramiento considerable en: tiempo, presupuesto y calidad (menor cantidad de defectos y mayor productividad).

- ***Tesis: Modelo de Elicitación de Requerimientos en Sistemas de Software construidos mediante el Proceso Unificado, Vásquez Lerzundi Wilfredo. Programa Profesional de Ingeniería de Sistemas. Universidad Católica de Santa María. 2008***

  Los trabajos de investigación actuales en ingeniería buscan mecanismos que permitan establecer la relación entre la funcionalidad esperada de un sistema de información y los procesos de negocios a los que éste dará soporte. Al crecer el tamaño y la complejidad de los sistemas, el desarrollo del mismo se divide en fases o etapas, siendo la ingeniería de requisitos la primera o del comienzo, evitar errores en esta etapa es primordial, ya que son los más costosos de reparar.

  La Elicitación consiste en las primeras actividades a realizarse en la Ingeniería de requisitos, aunque esta etapa no se puede “divorciar” de las demás, ya que seguramente se itereresará a través de las mismas durante el desarrollo de los requerimientos.

Este enfoque permitirá asegurar que el sistema de información a desarrollar sea realmente útil en las tareas de los actores organizacionales. Los trabajos de investigación en esta área han determinado que las metas organizacionales son una buena base para establecer la relación entre los objetivos perseguidos por el negocio y los requisitos del sistema de información a desarrollar, estos requisitos (funcionales y no funcionales) deben corresponderse con tareas que se desean desempeñar dentro de un proceso de negocios. Los procesos de negocio a su vez, permiten el cumplimiento o satisfacción de alguna o algunas de las metas del negocio.

Este trabajo permite tener un punto de partida sólido para la construcción del sistema de información, donde cada requisito tiene su origen en las metas del negocio.

- ***Tesis: Modelo de Auditoria Informática Aplicado a la Especificación de Requerimientos de Software de Sistemas de Información, Mamani Amanqui, Flor Karina. Programa Profesional de Ingeniería de Sistemas. Universidad Católica de Santa María. 2012***

  Básicamente es un aporte para la identificación y posterior corrección de vulnerabilidades que se pudieran presentarse en la obtención de requerimientos de software de los diversos sistemas de información.

  Tener un modelo de auditoria aplicada de manera adecuada asegura la integridad de los sistemas de información.

  La auditoría informática permite la revisión y evaluación de los sistemas de información son considerados como una evaluación cuyo fin es detectar errores y señalar fallas, sino también de evaluar la adecuada utilidad de la información. La recolección de requerimientos es una de las fases más importantes de la ingeniería

de software por lo que se determina cual es la naturaleza del producto que se va desarrollar para la empresa.

El trabajo tiene como fin desarrollar un modelo de auditoria informática aplicado a la generación de especificaciones de requerimientos.

- ***Tesis: Metodología para la Especificación de Requisitos de un Producto de Software de Calidad Bajo ISO 9000, Mamani Amanqui, Flor Karina. Programa Profesional de Ingeniería de Sistemas. Universidad Católica de Santa María. 2012***

  Una buena especificación de requisitos es el primer paso para asegurar la calidad del software. Este trabajo presenta una metodología para obtener una especificación de requisitos de software. La metodología ha sido desarrollada teniendo en cuenta las exigencias de la norma ISO 9000-3 y el estándar IEEE 830, consta de cuatro actividades las cuales permiten descubrir los requisitos del sistema de distribución y comercialización de la empresa consumo distribuciones S.A.C.

  En el mundo de los negocios del siglo XXI, la información es un insumo estratégico para la garantía de la competitividad; la tecnología de la información su principal herramienta y el software el factor que marca la diferencia.

  Desde que las computadoras pasaron a formar parte de nuestra rutina diaria, la producción de software ha tenido un aumento constante y cada vez más exigente, los usuarios del software requieren de productos que les atiendan en mayor grado sus necesidades, buscando productos con mayores prestaciones y además con un bajo costo; los usuarios desean productos de software de calidad.

  ISO 9000 define un sistema para asegurar la calidad del producto. Para que las empresas encargadas del desarrollo de software logren asegurar la calidad de sus productos, estas deben ajustar sus procesos a las exigencias establecida por el sistema

de calidad ISO 9001. Debido a la gran diferencia entre el software y cualquier otro producto. La (ISO) publico el estándar internacional ISO 9000-3 para guiarlas en la aplicación de ISO 9001.

ISO 9000-3 establece como una de sus principales exigencias que antes de proceder con el desarrollo de un producto de software, se debe tener un conjunto completo y sin ambigüedades de los requisitos del mismo. Ya que la calidad del software depende de una buena especificación de requisitos.

La determinación de los requisitos es una actividad extremadamente importante. Los requisitos representan las necesidades del cliente, además los requisitos son la base para el planeamiento, el desarrollo del proyecto, y proporciona los medios para valorar la calidad del producto de software. La determinación de los requerimientos puede parecer una tarea relativamente sencilla, pero no es así.

Dada la naturaleza del ISO 9000-3, no determina “COMO” obtener una especificación de requerimientos del software a desarrollar. Dejando a los interesados plantear su propio método. Y es ahí donde existe un vacío que creo es necesario llenar; este es el espíritu del trabajo de tesis. Conocer un conjunto de pasos que nos permitan especificar los requisitos del software de una manera metódica.

2.2. Ingeniería del Software

La ingeniería de software es una disciplina o área de la informática que ofrece métodos y técnicas, para desarrollar y mantener software de calidad que resuelven problemas de todo tipo. Hoy día es cada vez más frecuente la consideración de la Ingeniería de Software como una nueva área de la ingeniería y comienza a ser una profesión implantada en el mundo laboral con derechos, deberes y responsabilidades que cumplir (Pressman, 2005).

Esta disciplina trasciende la actividad de programación, que es el pilar fundamental a la hora de crear una aplicación. El Ingeniero de Software se encarga de toda la gestión del proyecto, para que éste se pueda desarrollar en un plazo determinado y con el presupuesto previsto.

La Ingeniería de Software, por lo tanto, incluye el análisis previo de la situación, el diseño del proyecto, el desarrollo del software, las pruebas necesarias para confirmar su correcto funcionamiento y la implementación del sistema.

Cabe destacar que el proceso de desarrollo de software implica lo que se conoce como ciclo de vida del software, que está formado por cuatro etapas: concepción, elaboración, construcción y transición.

La concepción fija el alcance del proyecto y desarrolla el modelo de negocio; la elaboración define el plan del proyecto, detalla las características y fundamenta la arquitectura; la construcción es el desarrollo del producto; y la transición es la transferencia del producto terminado a los usuarios.

Una vez que se completa este ciclo, entra en juego el mantenimiento del software. Se trata de una fase donde se solucionan los errores descubiertos (muchas veces advertidos por los propios usuarios) y se incorporan actualizaciones para hacer frente a los nuevos requisitos. El proceso de mantenimiento incorpora además nuevos desarrollos, para permitir que el software pueda cumplir con una mayor cantidad de tareas.

Un campo directamente relacionado con la Ingeniería de Software es la arquitectura de sistemas, que consiste en determinar y esquematizar la estructura general del proyecto, diagramando su esqueleto con un grado relativamente alto de especificidad y señalando los distintos componentes que serán necesarios para llevar a

cabo el desarrollo, tales como aplicaciones complementarias y bases de datos. Se trata de un punto fundamental del proceso, y es muchas veces la clave del éxito de un producto informático.

Los avances tecnológicos y su repercusión en la vida social han afectado inevitablemente el proceso de desarrollo de software por diversos motivos, como ser el acceso indiscriminado de los usuarios a cierta información, que hasta hace un par de décadas desconocía por completo y que no pueden comprender, dado que no poseen el grado de conocimiento técnico necesario. Un consumidor bien informado es un consumidor al que no se puede engañar, ya que sabe lo que necesita y tiene la capacidad de analizar las diferentes ofertas del mercado, comparando las propuestas y prestaciones de los productos; sin embargo, un consumidor mal informado es como un niño caprichoso que llora, grita y patalea sin parar.

La primera de todas las etapas del trabajo que realizan los ingenieros de software consiste en estudiar minuciosamente las características que se creen necesarias para el programa a desarrollar, y es éste el punto en el cual deben encontrar un equilibrio (cada vez más difícil de alcanzar) entre las demandas excesivas de los malos consumidores y las posibilidades de la compañía. El tiempo es dinero, y las empresas del mundo informático lo saben muy bien.

Cada función de un programa, cada rasgo que lo vuelva más cómodo, más inteligente, más accesible, se traduce en una cantidad determinada de tiempo, que a su vez acarrea los sueldos de todas las personas involucradas en su desarrollo. Pero además del costo de producción necesario para realizar cada una de las piezas de un programa, la ingeniería de software debe decidir cuáles de ellas tienen sentido, son coherentes con

el resto y son necesarias para comunicar claramente la esencia y los objetivos de la aplicación.

La Ingeniería del Software se ocupa de los siguientes temas:

- Desarrollo y mantenimiento de sistemas software:
- Sistemas eficientes y confiables y susceptibles de mantener.
- Problema: El software creció en tamaño y complejidad.
- Aplicaciones de seguridad críticas.
- Gran impacto de sistemas software enormes y costosos.
- Distinta a otras disciplinas de ingeniería.
- Naturaleza intangible y continua del software.

2.3. Procesos de Software

Un proceso de software es un conjunto de actividades y resultados asociados que producen un producto de software. Estas actividades son llevadas a cabo por los ingenieros de software (Sommerville, 2005).

Además, podemos decir que el concepto de proceso de software es uno de los conceptos más abstractos en la historia de la Ingeniería de Software. Muchas veces es confundido como un proyecto informático o como un abstracto que pretende ser la base de una potencial gestión de conocimiento de los procesos de desarrollo informático. Se ha logrado concebir una cosmovisión del proceso de software más formal, siendo fijada como un proceso de continuo aprendizaje mediante el cual una organización mejora y se mejora a través de procesos adquiridos y/o sus propios procesos. Así se llega al proceso llamado CMM o Capability Maturity Model con sus modelos asociados a personas y adquisiciones. La versión actualizada del CMM, lCMMI o CMM Integrated, es

actualmente un instrumento de la gestión de la ingeniería de software tradicional y formal.

No obstante, esta formalización ha hecho que su validez se pierda como "un proceso más a ejecutar" y no en su sentido organizador del trabajo informático sobre todos los artefactos informáticos que una organización utiliza hoy en día. Aún más, los adelantos en ingeniería organizacional y su pervivencia en las empresas hoy en día, hacen de que un enfoque de proceso de software no se limite ni constriña a un conjunto de prácticas y al cumplimiento de un modelo, sino a un estado o proceder organizacional tendiente a gestionar el conjunto de proyectos, procesos y tareas asociadas al procesamiento de datos, de información y de conocimiento de una organización cualquiera.

En este sentido se presenta el proceso de software bajo una óptica de enclave organizacional que precisa una gestión completa, cabal e integral. Con relación a la Ingeniería de Software como disciplina y al Ingeniero de Software como profesional, el enfoque dado al concepto de proceso de software resulta integrador con otros campos relacionados como la calidad y la gestión de riesgos, la gestión de proyectos y, la administración de sistemas tecnológicos al introducir conceptos de ventas versus diseño, selección de componentes o simplemente la constitución de una Oficina de Proyectos Informáticos.

En este sentido se introduce como término provocador el de proceso de negocio de software, para encaminar hacia esta nueva idea del alcance de un proceso de software, ya no como un conjunto de "cosas por hacer" para producir software, sino un conjunto de procesos claves y estratégicos que regulan todo el negocio adscrito a una producción de software lo cual no limita los procesos a la producción tradicional, sino

que incluye toda la cadena de valor que aporta valor a un software o desarrollo tecnológico cualquiera, lo cual incluye marketing, gerencia, investigación, etc.

2.4. Calidad de Software

La calidad del software engloba una serie de técnicas, métodos, modelos y guías de buenas prácticas que tienen como fin último garantizar la bondad de los resultados que se obtienen en los proyectos de desarrollo software. La búsqueda y aseguramiento de unos resultados de calidad en cualquier ingeniería, resulta necesaria y se debe contemplar de manera intrínseca en el desarrollo. Desarrollar un software de calidad va a requerir el seguimiento de una serie de estándares, de una serie de referentes en buenas prácticas y del uso de medidas objetivas que permitan observar, medir y controlar aspectos tanto cuantitativos como cualitativos de los resultados.

La necesidad de desarrollar el software bajo el paraguas de la calidad, está siendo un valor en alza que cada día se requiere más en la sociedad de una manera explícita y que es la garantía de éxito de los proyectos. Para lograr este fin, bajo el paraguas que se entiende por calidad del software es necesario incluir entornos metodológicos que, no solo se basen en estándares y modelos de referencia, sino que, además, permitan hacer un seguimiento objetivo y cuantificable de la calidad de los productos que van obteniendo así como la utilización de herramientas para el soporte del mismo.

Producto y Proceso: Garantía

- ¿Cómo puedo garantizar la calidad de un producto software?
  - El proceso: un *framework*
  - En cascada
  - Iterativo

- Incremental
- Verificación

- Concepto:
- ¿El producto se está construyendo en forma correcta?
  - El proceso de desarrollo debe estar conforme a estándares o prácticas de desarrollo.
  - Se verifica también la funcionalidad del software.
- ¿Cómo verificar el proceso?
- ¿Cómo verificar la funcionalidad?
- El software debe ajustarse a su especificación

También podemos decir que la calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad.

Todos estamos familiarizados con los problemas derivados de un fallo de funcionamiento en el sistema informático. Por alguna razón no obvia, a veces los sistemas informáticos caen y no consiguen realizar los servicios que se les ha requerido. Los programas que se ejecutan sobre dicho sistemas, pueden no funcionar como se esperaba y pueden corromper los datos que son gestionados por el sistema (Sommerville, 2005).

La calidad del software es medible y varía de un sistema a otro o de un programa a otro. Un software elaborado para el control de naves espaciales debe ser confiable al nivel de "cero fallas"; un software hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad; mientras que un producto de software para ser explotado

durante un largo período (10 años o más), necesita ser confiable, mantenible y flexible para disminuir los costos de mantenimiento y perfeccionamiento durante el tiempo de explotación.

La calidad del software puede medirse después de elaborado el producto. Pero esto puede resultar muy costoso si se detectan problemas deriva dos de imperfecciones en el diseño, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida del software.

¿Cómo obtener un software de calidad?

La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.

La política establecida debe estar sustentada sobre tres principios básicos: tecnológico, administrativo y ergonómico.

- El principio tecnológico define las técnicas a utilizar en el proceso de desarrollo del software.
- El principio administrativo contempla las funciones de planificación y control del desarrollo del software, así como la organización del ambiente o centro de ingeniería de software.
- El principio ergonómico define la interfaz entre el usuario y el ambiente automatizado.

La adopción de una buena política contribuye en gran medida a lograr la calidad del software, pero no la asegura. Para el aseguramiento de la calidad es necesario su control o evaluación.

¿Cómo controlar la calidad del software?

Para controlar la calidad del software es necesario, ante todo, definir los parámetros, indicadores o criterios de medición, ya que, como bien plantea Tom De Marco, "usted no puede controlar lo que no se puede medir".

Las cualidades para medir la calidad del software son definidas por innumerables autores, los cuales las denominan y agrupan de formas diferentes. Por ejemplo, John Wiley define métricas de calidad y criterios, donde cada métrica se obtiene a partir de combinaciones de los diferentes criterios. La Metodología para la evaluación de la calidad de los medios de programas de la CIC, de Rusia, define indicadores de calidad estructurados en cuatro niveles jerárquicos: factor, criterio, métrica, elemento de evaluación, donde cada nivel inferior contiene los indicadores que conforman el nivel precedente. Otros autores identifican la calidad con el nivel de complejidad del software y definen dos categorías de métricas: de complejidad de programa o código, y de complejidad de sistema o estructura.

Todos los autores coinciden en que el software posee determinados índices medibles que son las bases para la calidad, el control y el perfeccionamiento de la productividad.

Una vez seleccionados los índices de calidad, se debe establecer el proceso de control, que requiere los siguientes pasos:

- Definir el software que va a ser controlado: clasificación por tipo, esfera de aplicación, complejidad, etc., de acuerdo con los estándares establecidos para el

desarrollo del software.

- Seleccionar una medida que pueda ser aplicada al objeto de control. Para cada clase de software es necesario definir los indicadores y sus magnitudes.
- Crear o determinar los métodos de valoración de los indicadores: métodos manuales como cuestionarios o encuestas estándares para la medición de criterios periciales y herramientas automatizadas para medir los criterios de cálculo.
- Definir las regulaciones organizativas para realizar el control: quiénes participan en el control de la calidad, cuándo se realiza, qué documentos deben ser revisados y elaborados, etc.

A partir del análisis de todo lo anterior, es necesario un proyecto para el Aseguramiento de la Calidad del Software (ACS), válido para cualquier entidad que se dedique a la investigación, producción y comercialización del software, el cual incluye la elaboración de un sistema de indicadores de la calidad del software, la confección de una metodología para el aseguramiento de la calidad del software y el desarrollo de herramientas manuales y automatizadas de apoyo para la aplicación de las técnicas y procedimientos del ACS, de forma tal que se conforme un sistema de aseguramiento de la calidad del software.

2.5. Aseguramiento de la Calidad

El aseguramiento de calidad (*Quality Assurance* – QA) de proceso y productos software, ofrece un conjunto de prácticas fundamentadas en normas de calidad internacionales, cuyo foco radica en sembrar en los proyectos la confianza de que sus productos satisfacen los requisitos de calidad que esperan percibir los usuarios.

Se fundamenta en la premisa de que el impacto de un defecto sobre los costos de un proyecto de desarrollo software será menor entre más temprana sea su detección, con base en esta premisa le ofrecemos a nuestros clientes un acompañamiento sistemático e integral para minimizar el riesgo de la ocurrencia de defectos en producción, fase en la que el defecto impacta no sólo a nivel monetario sino también en la imagen corporativa de nuestros clientes. Claramente los intangibles como el *time to market* y la satisfacción son los obtenibles de este proceso.

Actualmente se apuesta a la adopción del modelo de referencia ISTQB (*International Software Testing Qualifications Board*) el cual brinda un esquema de profesionalización del perfil del Analista de Calidad. A través de las certificaciones que ofrece el ISTQB, se busca motivar en los analistas el sentido de mejoramiento continuo de "la forma de realizar el *testing* de productos software" y que vean en el *testing* una carrera profesional donde se fijan metas a alcanzar las cuales se logran capitalizando la experiencia que se gana en cada uno de los proyectos. En este camino de la profesionalización del rol del analista de calidad de software es acompañar a todos los clientes, por lo cual los proyectos generan espacios de transferencia de conocimiento y crecimiento compartido y bidireccional que sin duda redunda en equipos de trabajo con altos niveles de motivación y performance.

Los servicios más demandados y con mayor proyección, son:

a. Pruebas funcionales y de regresión automatizadas.
b. Pruebas funcionales manuales.
c. Homologación
d. Pruebas de rendimiento (performance).
e. Pruebas de UI (*User Interface* – interfaz de usuario).

2.6. Pruebas de Software

El desarrollo de software implica la realización de una serie de actividades predispuestas a incorporar errores (en la etapa de definición de requerimientos, de diseño, de desarrollo, etc.). Por ejemplo, Sommerville (2005) indica que los requerimientos funcionales son declaraciones de los servicios que debe proporcionar el sistema de la manera en que este debe de reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares

Debido a que estos errores se deben a la habilidad innata de provocar errores, se tiene que incorporar una actividad que garantice la calidad del software.

En la etapa de prueba del software se crean una serie de casos de prueba que intentan "destruir" el software desarrollado.

La prueba requiere que se descarten ideas preconcebidas sobre la "calidad o corrección" del software desarrollado.

El proceso de prueba tiene dos entradas:

- Configuración del software: incluye la especificación de requisitos del software, la especificación del diseño y el código fuente
- Configuración de prueba: incluye un plan y un procedimiento de prueba

Si el funcionamiento del software parece ser correcto y los errores encontrados son fáciles de corregir, se concluye que:

- La calidad y la fiabilidad del software son aceptables, o que
- Las pruebas son inadecuadas para descubrir errores serios

2.7. Diseño de Casos de Prueba

“Existe una sola regla para el diseño de casos de prueba: cubrir todas las posibilidades, sin hacer demasiados casos de prueba” Tsuneo Yamaura (Pressman,

2005). Se trata de diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y de tiempo.

Cualquier producto de ingeniería se puede probar de dos formas:

- Pruebas de caja negra: Realizar pruebas de forma que se compruebe que cada función es operativa.
- Pruebas de caja blanca: Desarrollar pruebas de forma que se asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada.

En pruebas de caja negra, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.

En pruebas de caja blanca se realiza un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del programa en varios puntos.

A primera vista, la prueba de caja blanca profunda nos llevaría a tener "programas cien por cien correctos", es decir:

- Definir todos los caminos lógicos
- Desarrollar casos de prueba para todos los caminos lógicos
- Evaluar los resultados

Pero esto supone un estudio demasiado exhaustivo, que prolongaría excesivamente los planes de desarrollo del software, por lo que se hará un estudio de los caminos lógicos importantes.

2.7.1. Pruebas de Caja Negra

Denominadas también *Pruebas de Comportamiento,* se centran en los requisitos funcionales del software. O sea, la prueba de caja negra permite al Ingeniero de Software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (Pressman, 2005).

Las pruebas de caja negra se llevan a cabo sobre la interface del software, obviando el comportamiento interno y la estructura del programa.

Los casos de prueba de caja negra pretenden demostrar que:

- Las funciones del software son operativas
- La entrada se acepta de forma correcta
- Se produce una salida correcta
- La integridad de la información externa se mantiene

A continuación se derivan conjuntos de condiciones de entrada que utilicen todos los requisitos funcionales de un programa.

Las pruebas de caja negra pretenden encontrar estos tipos de errores:

- Funciones incorrectas o ausentes
- Errores en la interface
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación

Los tipos de prueba de caja negra que se aplicaron fueron:

- Prueba de partición equivalente
- Prueba de análisis de valores límites
- Pruebas de Partición Equivalente

Un buen caso de prueba seria aquel que tiene una razonable probabilidad de encontrar un error. Por otro lado, en la práctica es casi imposible determinar pruebas con entradas exhaustivas para un programa. Por lo tanto, en pruebas de software, estamos limitados a tratar con un pequeño subconjunto de todas las posibles entradas.

Por supuesto, se desea seleccionar un subconjunto de entradas correctas, un subconjunto con la más alta probabilidad de encontrar la mayor cantidad de errores.

Una forma de localizar a este subconjunto es verificar que un caso de prueba bien seleccionado, debe tener también otras dos propiedades:

- Reducir o simplificar el número de casos de prueba que deben desarrollarse para lograr algún objetivo predefinido de prueba "razonable".
- Cubrir un amplio conjunto de otros posibles casos de prueba. Es decir, se dice algo acerca de la presencia o ausencia de errores por encima de este conjunto específico de valores de entrada.

Estas dos propiedades, a pesar de que parecen ser similares, describen dos consideraciones distintas. La primera implica que cada caso de prueba debe invocar muchas consideraciones de entrada diferentes, como sea posible para minimizar el número total de casos de prueba necesarios. El segundo implica que usted debe tratar de dividir el dominio de entrada de un programa en un número finito de clases de equivalencia de tal manera que usted pueda razonablemente suponer (aunque, por supuesto, no estar absolutamente seguro) de que una prueba de un valor representativo de cada clase es equivalente a una prueba de cualquier otro valor.

Es decir, si un caso de prueba en una clase de equivalencia detecta un error, todos los otros casos de prueba, en la clase de equivalencia, se espera que encuentren el mismo error. Por el contrario, si un caso de prueba no detectó un error, es de esperar que

no haya otros casos de prueba en la clase de equivalencia, sino que caigan dentro de otra clase de equivalencia, ya que las clases de equivalencia pueden superponerse entre sí.

Estas dos consideraciones de la metodología de caja negra constituyen una partición de equivalencia. La segunda consideración se utiliza para desarrollar un conjunto "interesante" de condiciones para ser probadas. La primera consideración es desarrollar un conjunto mínimo de casos de prueba que cubra estas condiciones.

Un ejemplo de una clase de equivalencia para un programa cuyo objetivo es determinar un tipo de triángulo en base a tres entradas enteras (a, b y c), sería el conjunto de "tres números enteros que tienen valores iguales mayores que cero." Al identificar esto como una clase de equivalencia, estamos afirmando que si no hay error se encuentra en una prueba de un elemento del conjunto, es poco probable que un error se encuentre con una prueba de otro elemento del conjunto. En otras palabras, nuestro tiempo de prueba es mejor invertido en diseñar otros casos de prueba (en diferentes clases de equivalencia).

Para diseñar un caso de prueba en base a la técnica de clases de equivalencia se procede en dos pasos:

- Identificar las clases de equivalencia
- Definir los casos de prueba.

Identificar las Clases de Equivalencia

Las clases de equivalencia se identifican mediante la adopción de cada condición de entrada o salida (generalmente una frase o sentencia en la especificación) y se divide en 1 o más grupos. Se identifican dos tipos de clase de equivalencias: validas e invalidas.

Dada una entrada o una condición externa, la identificación de las clases de equivalencia es en gran parte un proceso heurístico. Un conjunto de directrices siguiente:

- Rango de Valores: Si una condición de entrada especifica un rango de valores (por ejemplo, “el número de elementos de 1 a 99").
- Número de Valores: Si una condición de entrada específica el número de valores (uno a seis propietarios por automóvil).
- Conjunto de Valores: Si una condición de entrada especifica un conjunto de valores de entrada y no hay razón para creer que el programa maneja cada uno de manera diferente (“Tipo de vehículo debe ser autobuses, camiones, taxi, motocicleta”), identificar una clase válida de equivalencia para cada uno y una clase de equivalencia no válida (por ejemplo “tráiler”).
- Debe ser: Si una condición de entrada especifica una situación "deber ser", tales como "el primer carácter del identificador debe ser una letra," identificar una clase de equivalencia válida (se trata de una letra) y una clase de equivalencia no válida (que no sea una letra).

Si hay alguna razón para creer que ciertos elementos de una clase no son tratados de forma idéntica, dividir la clase de equivalencia en pequeñas clases de equivalencia.

La Identificación de los Casos de Prueba

El segundo paso es el uso de clases de equivalencia para identificar los casos de prueba. El proceso es el siguiente:

- Asignar un número único a cada clase de equivalencia.

  Escribir casos de prueba hasta que todas las clases de equivalencia hayan sido cubiertas, intentando cubrir en cada caso tantas clases de equivalencia validas

como sea posible.

Para cada clase de equivalencia invalida única no cubierta, escribir un caso de prueba, hasta que todas las clases de equivalencia invalidas hayan sido cubiertas.

La razón de que los casos de prueba individuales cubren los casos no válidos, es que ciertos controles de entrada errónea revisen la máscara o remplacen otros controles de error de ingreso de usuario. Por ejemplo, si la especificación indica "entrar tipo de libro (tapa dura, tapa blanda o suelta) y cantidad (1-999),"el caso de prueba, XYZ 0, expresa dos condiciones de error (tipo libro no válida y cantidad, probablemente no sea necesario revisar la cantidad, ya que el programa puede decir "XYZ es el tipo de libro desconocido" y no sea necesario examinar el resto de la entrada.

Un ejemplo

Una empresa del sector financiero establece sus reglas de cálculo de tasas de interés de préstamo considerando lo siguiente: Tipo de Moneda, Monto y Plazo.

Si el monto del préstamo esta entre 5,000 y 10,000 se aplica el 9.5% de interés, pero si esta entre 10,001 y 25,000 se aplica el 9.0%. Si el préstamo es en Soles se incrementa la tasa en 0.5%, si es en Dólares la tasa aumenta en 1.5% y si es en Euros en 2.5%. Adicionalmente, el préstamo se puede otorgar entre 12 y 48 meses siendo la regla para el incremento; 0.01*(48-mes) %.

Es posible que desee considerar la forma en que estos casos de prueba se comparan con un conjunto de casos de prueba derivados de una manera ad hoc.

A pesar que partición de equivalencia es muy superior a una selección aleatoria de casos de prueba, aún tiene deficiencias. Pasa por alto ciertos tipos, como los casos de prueba de alto rendimiento, por ejemplo.

Análisis del Valor Límite

La experiencia demuestra que los casos de prueba que exploran las condiciones de frontera tienen una rentabilidad mayor que los casos de prueba que no lo hacen. Condiciones de frontera son aquellas situaciones que se dan directamente sobre y debajo de los bordes de entrada y salida de las clases de equivalencia. El análisis de los valores de frontera difiere de la partición de equivalencia en dos aspectos:

- En lugar de seleccionar cualquier elemento de una clase de equivalencia como representante, el análisis de valores en la frontera requiere que uno o más elementos se seleccionen de manera que cada borde de la clase de equivalencia sea objeto de una prueba.
- En lugar de simplemente centrar la atención en las condiciones de entrada (el espacio de entrada), los casos de prueba también se derivan de considerar el espacio de resultado (salida de clases de equivalencia).

Es difícil presentar un "recetario" para el análisis de valores en la frontera, ya que requiere un cierto grado de creatividad y un cierto grado de especialización hacia el problema en cuestión. (Por lo tanto, al igual que muchos otros aspectos de prueba, es más un estado mental que otra cosa), no obstante, algunas pautas generales pueden ser las siguientes:

- Si una condición de entrada especifica un rango de valores, escribir casos de prueba con valores válidos con los extremos, y casos de prueba no válidos de entrada para las situaciones más allá de los extremos. Por ejemplo, si el dominio de validez de un valor de entrada es -1.0 - +1.0, escribir casos de prueba para las situaciones -1.0, 1.0, -1.001 y 1.001.

- Si una condición de entrada especifica un número de valores, escribir casos de prueba para el número máximo y mínimo de valores, una por debajo y más allá de estos valores. Por ejemplo, si un archivo de entrada puede contener 1 a 255 registros, escribir casos de prueba de 0, 1, 255, y 256 registros.

Use la directriz 1 para cada condición de salida. Por ejemplo, si un programa calcula la deducción mensual de un salario, si el mínimo es S/. 0.00 y el máximo es de S/. 1,165.25, escribir casos de prueba que utilicen valores de S/. 0,00 y S/. 1,165.25 para ser deducidos. También, ver si es posible inventar casos de prueba que podrían causar una deducción negativa o una deducción de más de S/. 1,165.25.Tenga en cuenta que es importante examinar los límites del espacio de resultado, ya que no siempre es el caso de que los límites de los dominios de entrada representan el mismo conjunto de circunstancias, como los límites de los rangos de salida (por ejemplo, considerar una subrutina seno). Además, no siempre es posible generar un resultado fuera del rango de salida, no obstante, vale la pena considerar la posibilidad.

Use la directriz 2 para cada condición de salida. Si un sistema de recuperación de información muestra los resúmenes más relevantes sobre la base de una solicitud de entrada, pero nunca más de cuatro resúmenes, escribir casos de prueba de tal manera que el programa muestre cero, uno, y los cuatro resúmenes, y escribir un caso de prueba que podría hacer que el programa erróneamente muestre cinco resúmenes.

Si la entrada o salida de un programa es un conjunto ordenado (un archivo secuencial, por ejemplo, o una lista lineal o una tabla), centrar la atención en el primer y último elemento del conjunto.

Además, use su ingenio para buscar otras condiciones de frontera.

En el programa de análisis de un triángulo se puede ilustrar la necesidad de un análisis de valor de límite. Para los valores de entradas para representar un triángulo, que deben ser enteros mayores que 0, donde la suma de los dos es mayor que el tercero. Si se definen las particiones equivalentes, se puede definir una donde se cumple esta condición, y otro en el que la suma de dos de los enteros no es mayor que el tercero. Por lo tanto, dos posibles casos de prueba podrían ser 3-4-5 y 1-2-4. Sin embargo, hemos perdido un error probable. Es decir, si una expresión en el programa se codificara como A + B> = C en lugar de A + B> C, el programa erróneamente nos diría que el 1-2-3 representa un triángulo escaleno válido. Por lo tanto, la diferencia importante entre el análisis de valores en la frontera y la partición de equivalencia es que el análisis de valor de límite explora situaciones en y alrededor de los bordes de las particiones de equivalencia.

2.7.2. Caja Blanca (testeo estructural)

El método del camino básico (Caja Blanca), permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución (Pressman, 2005).

Analiza por módulos, testea lo que el programa hace. Por lo tanto, es necesario tener información acerca de la estructura interna del software, no es tan importante tener presentes las especificaciones del módulo. Se testea en base al conocimiento de la lógica del sistema y en base al estudio de la estructura del algoritmo.

Se comprueban los caminos lógicos del software, diseñando casos de prueba que ejerciten conjuntos de condiciones y/o bucles.

Técnicas de caja blanca

Prueba del camino básico: Se puede aplicar a un diseño procedimental detallado o a un código fuente. Los casos de prueba obtenidos garantizan que se ejecuta por lo menos una vez cada sentencia. Los pasos a seguir son:

a) Elaborar el Grafo de flujo.

b) Determinar la complejidad ciclomática

c) Determinar un conjunto básico de caminos linealmente independientes.

d) Preparar casos de prueba que ejecutan todos los caminos del punto c.

Complejidad ciclomática: Es una métrica de software que proporciona una medición a la complejidad lógica de un programa. Define el número de caminos independientes del conjunto básico de un programa. Establece el límite superior de pruebas a realizar para asegurar que cada sentencia se ejecuta por lo menos una vez.

Prueba de bucles: Se centra en la validez de las construcciones de bucles en un programa. Dentro de los bucles tenemos cuatro clases:

- Bucles simples: Siendo n el número de pasos máximo que el bucle permite, el conjunto de pruebas a aplicar será: saltar totalmente el bucle (la condición no se cumple, el bucle no se ejecuta), pasar una vez por el bucle, pasar dos veces por el bucle, hacer m pasos por el bucle con m < n, hacer n-1, n y n+1 pasos por el bucle.
- Bucles anidados: El conjunto de pruebas sugerido es: empezar en el bucle más interior y disponer de los demás bucles en sus valores mínimos; pruebas de bucles simples con este bucle interior; progresión hacia fuera, llevando a cabo pruebas para el siguiente bucle y manteniendo los bucles exteriores con sus valores mínimos; estos pasos se repiten hasta haber probado todos los bucles.
- Bucles concatenados: Si cada uno de los bucles es independiente, se pueden

seguir los pasos descriptos para los bucles simples, de lo contrario, se seguirán los pasos sugeridos para los bucles anidados.

- Bucles no estructurados: siempre que sea posible, esta clase de bucles se debe rediseñar para que se ajuste a las construcciones de programación estructurada.

Para cada tipo de bucles tenemos un conjunto de pruebas adicionales además del análisis del camino básico (que aísla los caminos de un bucle).

Prueba de condición:

Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. Estas condiciones pueden ser simples o compuestas.

– Condición simple: es una variable lógica o una expresión relacional, posiblemente precedida con un operador NOT.

– Condición compuesta: está formada por dos o más condiciones simples, operadores lógicos y paréntesis. (operadores lógicos permitidos: OR, AND, NOT)

Caja Negra (testeo funcional)

A diferencia de caja blanca, testea cómo deben comportarse los módulos. Por lo que se basa en los requerimientos del módulo, no en su estructura interna. Las pruebas se realizan sobre las interfaces y busca que las funciones sean operativas, que se obtenga una salida prevista para una entrada. Otro nombre para éstas es pruebas funcionales debido a que al probador sólo le interesa la funcionalidad y no la implementación del software.

El probador introduce las entradas en los componentes del sistema y examina las salidas correspondientes. Si las salidas no son las previstas, entonces la prueba ha detectado exitosamente un problema con el software.

CAPÍTULO III

MÉTODO PARA REALIZAR PRUEBAS DE SOFTWARE

3.1. Introducción

En este capítulo se verá los diferentes procesos para desarrollar el método de pruebas de acuerdo al proceso general del ciclo de vida del software. Se probará las aplicaciones seleccionadas para la implementación de los distintos tipos de pruebas de software de acuerdo al rol que cumple cada participante del proceso para la generación del método.

3.2. Ciclo de Vida del Software.

El método se desarrolla, de acuerdo a las fases mencionadas en el gráfico anterior.

3.3. Flujo General del proceso

Los participantes se detallan a continuación cada uno de ellos cumple un rol dentro de todo el proceso pero para la realización del método sus roles se definen en la sección siguiente.

- *Usuario Final:*

Se le denomina Usuario Final al área o persona dentro de la institución, la cual genera la idea y realiza la solicitud inicial de manera formal al área de T.I. para que se pueda ejecutar, este usuario sólo mantiene comunicación con el Analista Funcional.

El Usuario Final, realiza peticiones al área de T.I. (requerimientos) de diferentes tipos, estos son:

- Ideas de mejora continua.
- Requerimiento de normativa SBS.
- Correcciones de errores encontrados en el software actual.
- Mantenimiento para optimizar procesos en el software actual.
- Requerimiento de nuevos productos.
- Requerimiento de proyectos en el entorno del giro de negocio.

- *Analista Funcional:*

El Analista Funcional, se les nombra de esta manera a las personas o persona encargada de la comunicación entre el Usuario Final y el área de T.I. llámese “vínculo” entre estas dos áreas; dentro de sus funciones, se encuentran:

- Realizar el análisis funcional de las nuevas aplicaciones para la organización, así como actualizar o mejorar las que ya existen.

- Realizar la solicitud formal especificando los detalles de los requisitos tomados del Usuario Final y también especificando los detalles a desarrollarse como sistema, es decir, controlar, analizar y supervisar el desarrollo funcional de las aplicaciones, asegurando el correcto funcionamiento.

- *Comité de Aprobación:*

Es el comité donde se reúnen a decidir la prioridad del requisito solicitado por el Usuario Final a través del Analista Funcional. Dentro de este comité, el Analista Funcional expone la propuesta planteada dando solución al requisito del Usuario Final. Tal como se indica, el comité decide si es viable o no; se encuentra conformado por las entidades como jefaturas de las áreas dentro de T.I.

- *Desarrollador:*

- Es la persona quien recibe la especificación del requerimiento de acuerdo a lo solicitado y aprobado, sus funciones se detallan a continuación:
- Realizar el análisis y diseño del requerimiento.
- Implementar la propuesta.
- Realizar las pruebas que le corresponde luego de su implementación.
- Una vez, que el desarrollador haya implementado la solución propuesta, se inicia la etapa de pruebas correspondiente.

- *Coordinador:*

El coordinador, es la persona encargada de supervisar el trabajo del desarrollador como también revisar y realizar el análisis del requerimiento entregado de manera más óptima para que el desarrollador tenga claro lo que se solicita.

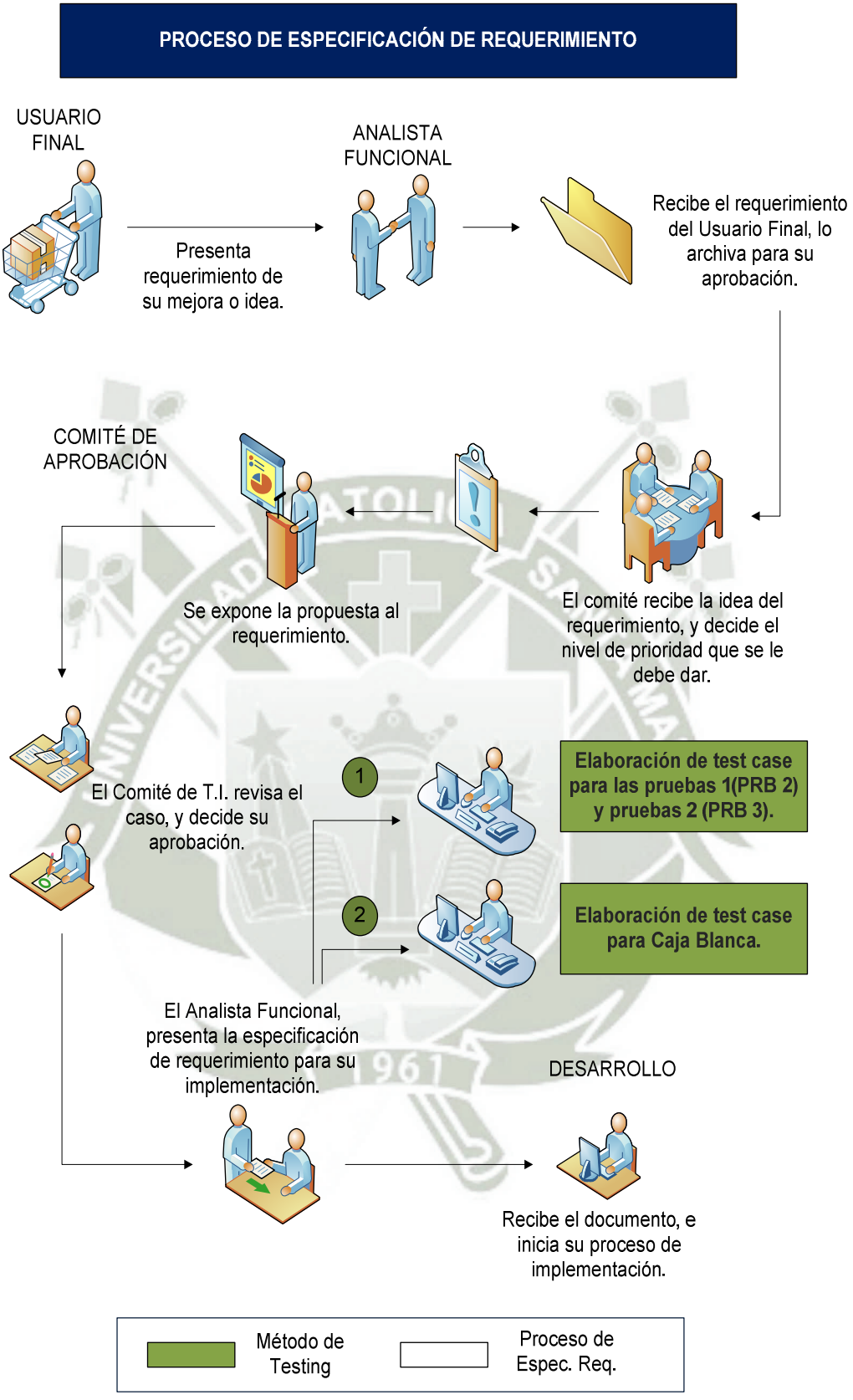

Figura 3.2: Proceso de especificación del requerimiento y aprobación

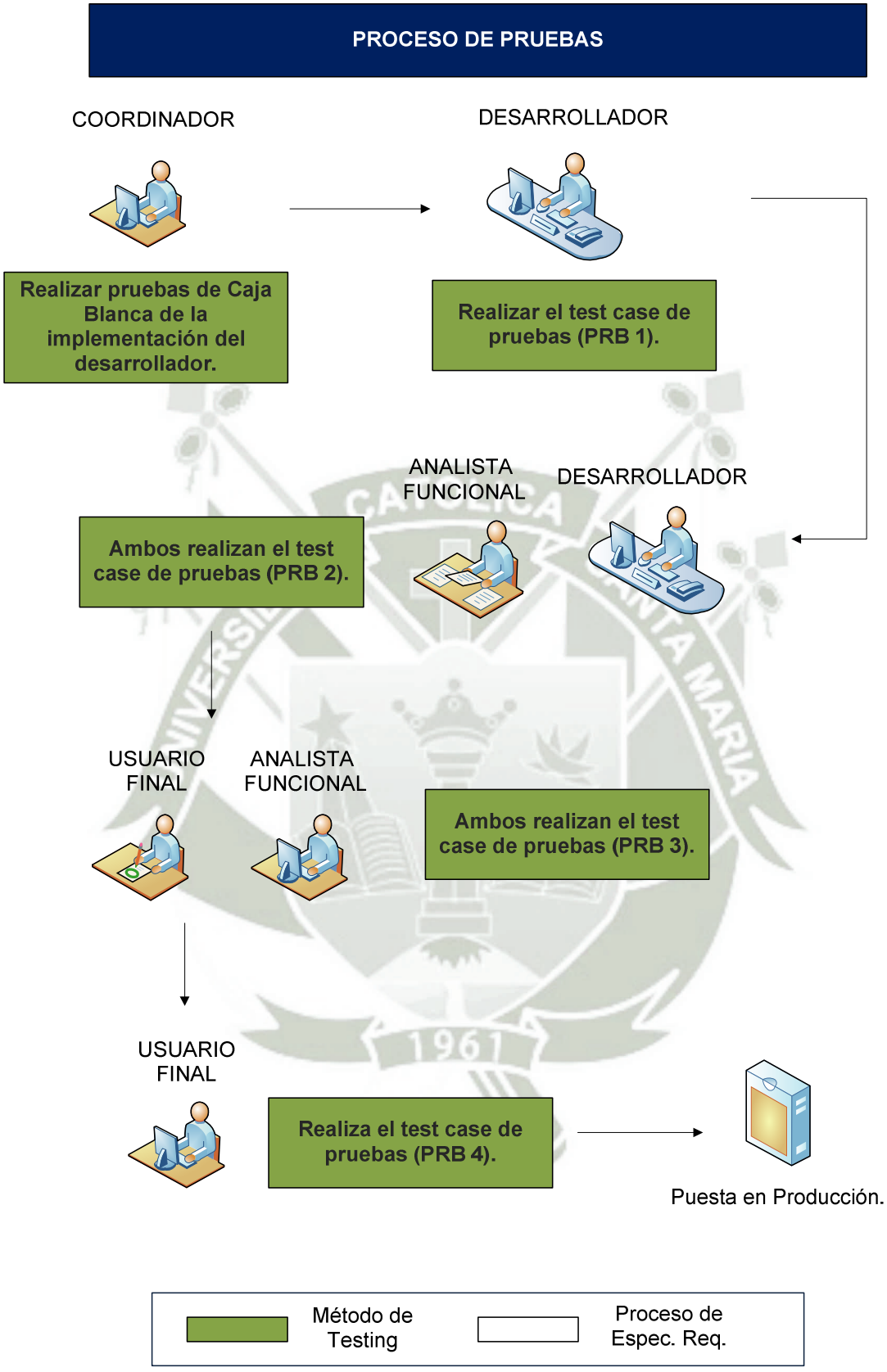

Figura 3.3: Proceso de Método de Pruebas de Software

3.4. Flujo del Proceso de Pruebas del Analista funcional.

3.4.1. Proceso

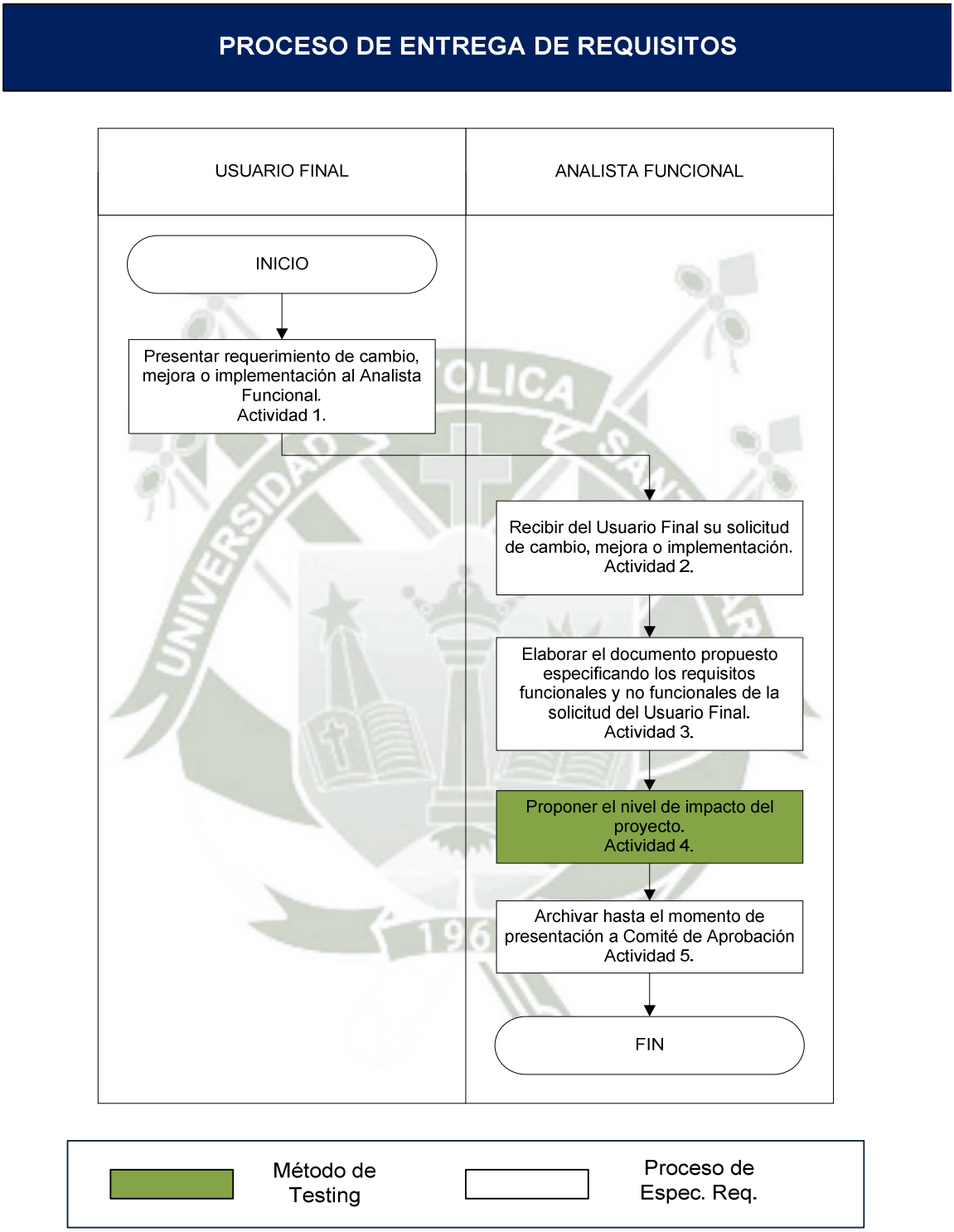

Figura 3.4: Proceso de prueba de parte del Analista Funcional.

3.4.2. Metodología

a) Generación de Test Case mediante CASOS DE USOS – Caja Negra

Los Casos de Usos y sus diagramas, facilitan la generación de casos de prueba para el tipo de caja negra, para verificar los requisitos funcionales solicitados para el sistema.

El Analista Funcional, al momento de recibir la solicitud de requerimiento por parte del Usuario Final, valida los requisitos no funcionales; a partir de estos, realiza la generación de Casos de Uso.

Una vez aprobada la propuesta, construye los casos de prueba de cada Caso de Uso, obtiene un objetivo del caso de donde se tomara como un caso de prueba. Teniendo su matriz de Casos de Uso, se elabora un diagrama de actividades para que se recorra al menos una vez el caso de uso, al generar el diagrama de actividades, se obtiene el objetivo generado por los casos de uso, y se identifica las variables y las combinaciones. Una vez ya generado el diagrama de actividades, se construye un mapa de dependencias o diagrama de dependencias, el cual examina todos los posibles caminos del diagrama de actividades. Y por último se definen las plantillas, con los casos de pruebas generados, de acuerdo al mapa de dependencias.

Al momento de diseñar los Casos de Uso, se va a definir los actores que serán los que interactúan con el sistema, también se define los eventos de cada actor. Estos actores trabajan en un escenario que también será identificado en los Casos de Uso, el cual determinara las entradas, salidas y límites que se tendrá en el sistema. Al desarrollar el diagrama de actividades, se contemplará los escenarios de los actores y se ordenara por su prioridad, de acuerdo al orden en que se trabaje en el módulo a probar y que cubra toda la funcionalidad del sistema. Seguido en el mapa de dependencias, se

escribe todos los caminos de los escenarios encontrados. Luego, se revisa el flujo principal del requerimiento y compara los escenarios faltantes; revisa todos los diagramas incluyendo el flujo principal, y arma la plantilla de los casos de prueba, para de esta manera contemplar todas las posibles contingencias.

b) Aplicación del Diagrama de Actividades en Caja Negra

En el diagrama de actividad, se refleja las actividades por donde un objeto pasa de estado en estado, y estos estados son en acción, indicando la acción al ejecutarse cuando pasa el objeto por él. Al generarse los escenarios partiendo por el caso de uso, se llega a completar por todos los estados, y es así que se recorre todo el camino, como cada caso de uso representa un caso de prueba, por ende existe más de un camino, y de igual manera el número de escenarios encontrados; esto dará que se recorra por lo menos una vez el camino trazado por el diagrama de actividades, validando el caso de uso junto al escenario.

Para completar con el diseño de pruebas, una vez obtenido el Diagrama de Actividades, se usara una mezcla de técnicas de pruebas de Caja Negra, para así comprobar cada caso específico con todos los valores; estas técnicas son:

- Particiones de equivalencia; el diagrama de actividades nos ayudará a definir cada camino como un identificador a cada clase, y los escenarios nos ayuda a clasificar las clases equivalentes válidas y las clases equivalentes no válidas.
- Valores límite; cada caso utilizando particiones de equivalencia, se ejercitará todos los valores límite que en el escenario se puede mostrar. Cada caso de uso, supone entrada de datos, donde se colocará los límites

c) Desarrollo de Mapa de Dependencia

Obtenidos los valores de acuerdo a las técnicas realizadas en el punto anterior, se desarrolla una matriz, en esta se colocan las clases, ya sean válidas o no válidas, y los valores establecidos en las entradas (rango), estos datos se fusionan para obtener el caso de prueba con su respectivo valor. Los valores de esta matriz, serán 0 o 1, de acuerdo a la funcionalidad de acuerdo a las entradas. El mapa debe de examinar cada caso que se haya generado en el diagrama de actividades; identificando también las restricciones de cada caso.

d) Definir Plantillas: Formatos según estándares para casos de pruebas.

Se crea un formato, indicando el caso de uso del cual se partió y los valores hallados en el mapa de dependencia, esta plantilla definida por un formato, describe el caso de prueba quedando listo para ser probado y luego completar con las observaciones de la persona encargada de la prueba.

3.5. Flujo del Proceso del Comité de Aprobación con el Desarrollador.

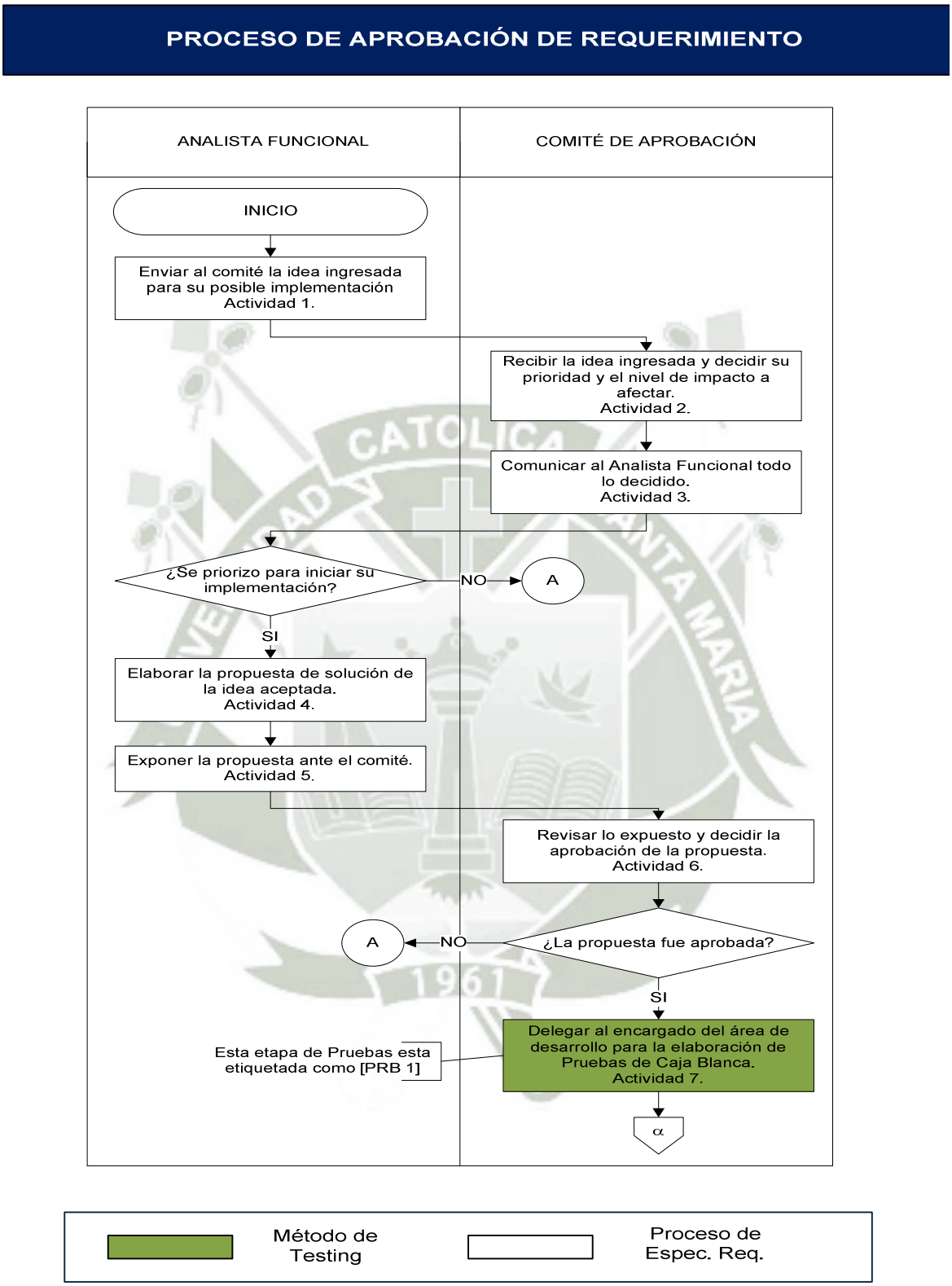

Figura 3.5: Proceso de aprobación del requerimiento

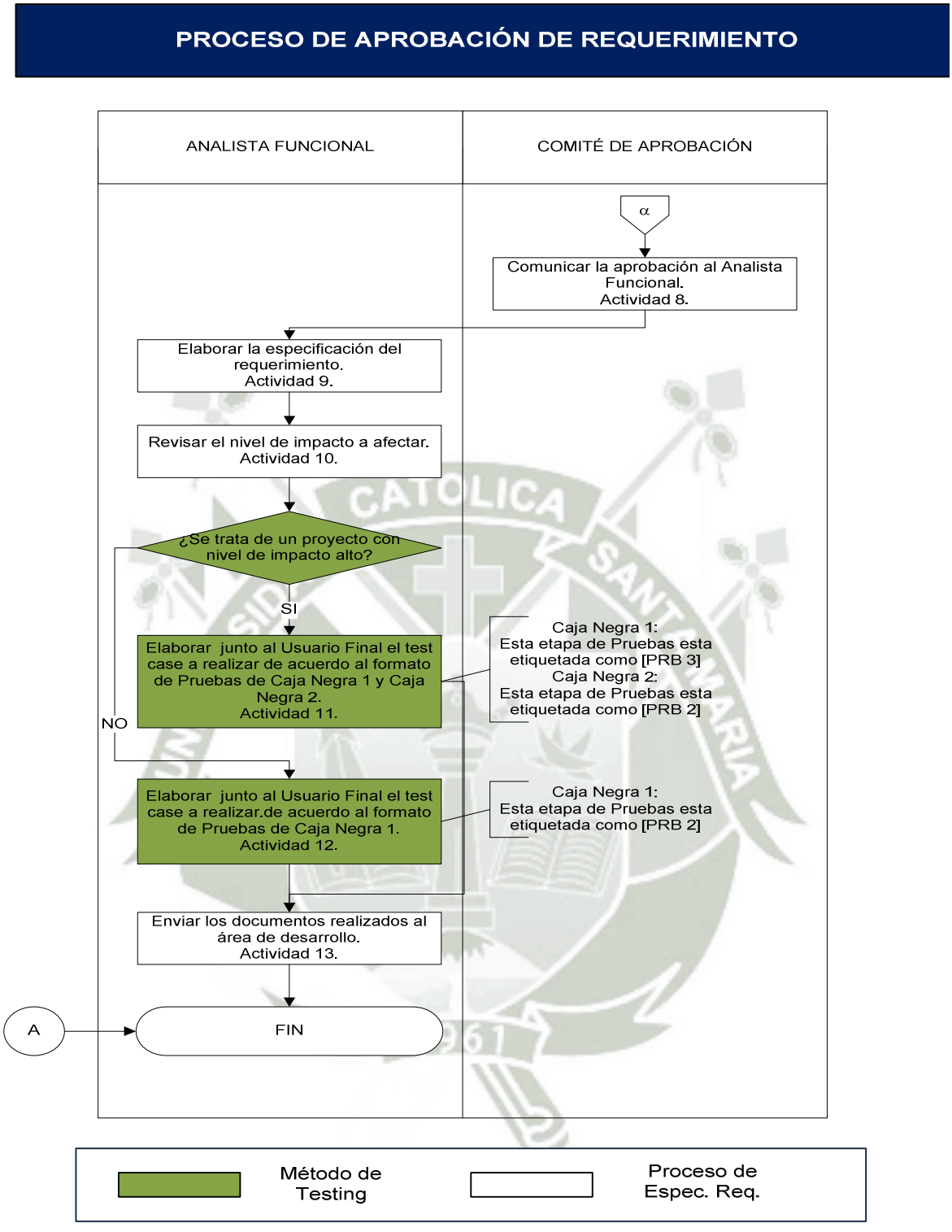

Figura 3.6: Continuación del proceso de aprobación del requerimiento

3.6. Flujo del Proceso de Testing del Desarrollador con el Analista Funcional y el Usuario Final.

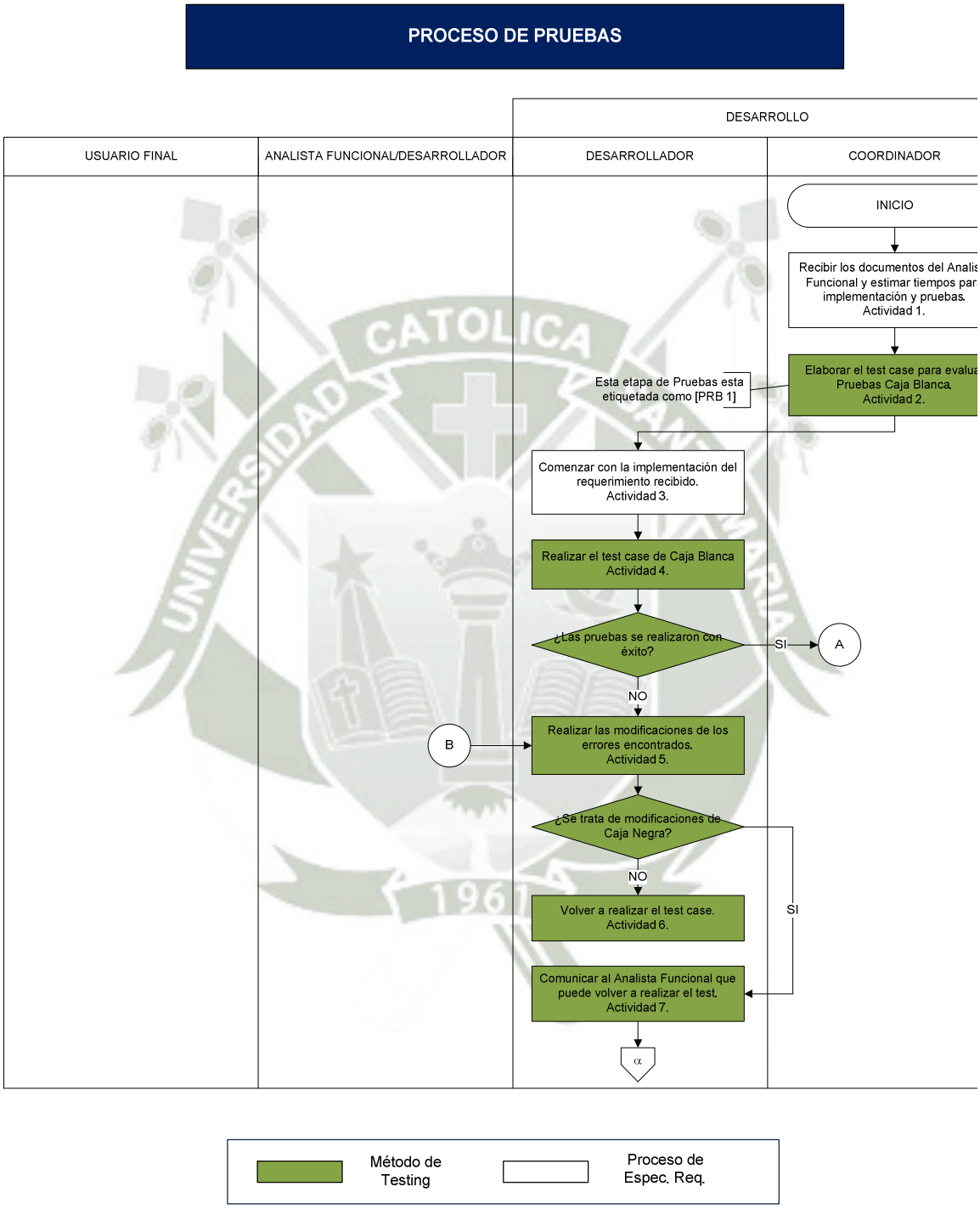

Figura 3.7: Proceso de Pruebas del Desarrollador con el Analista Funcional y el Usuario Final

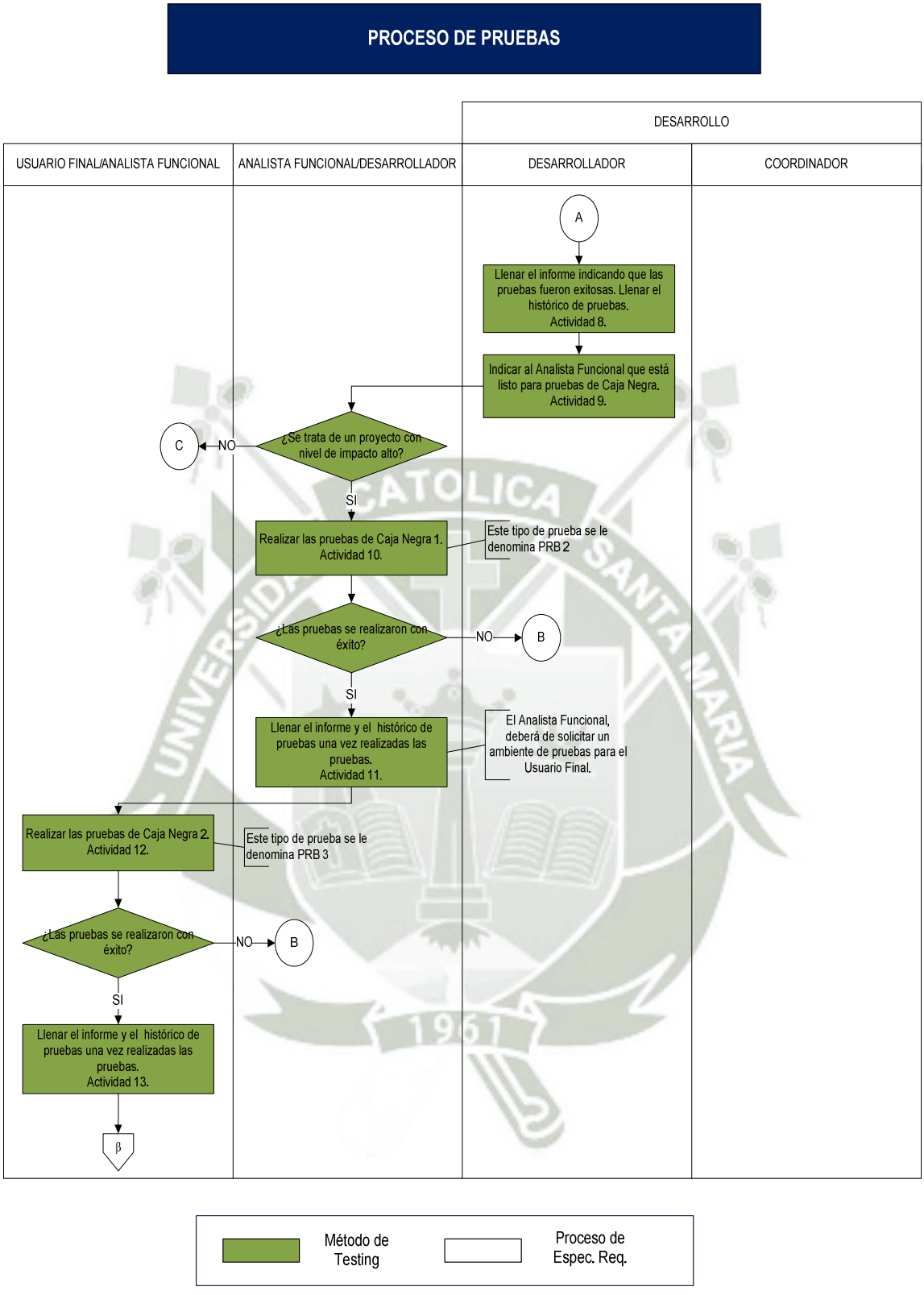

Figura 3.8: Continuación del Proceso de Pruebas del Desarrollador con el Analista Funcional y el Usuario Final

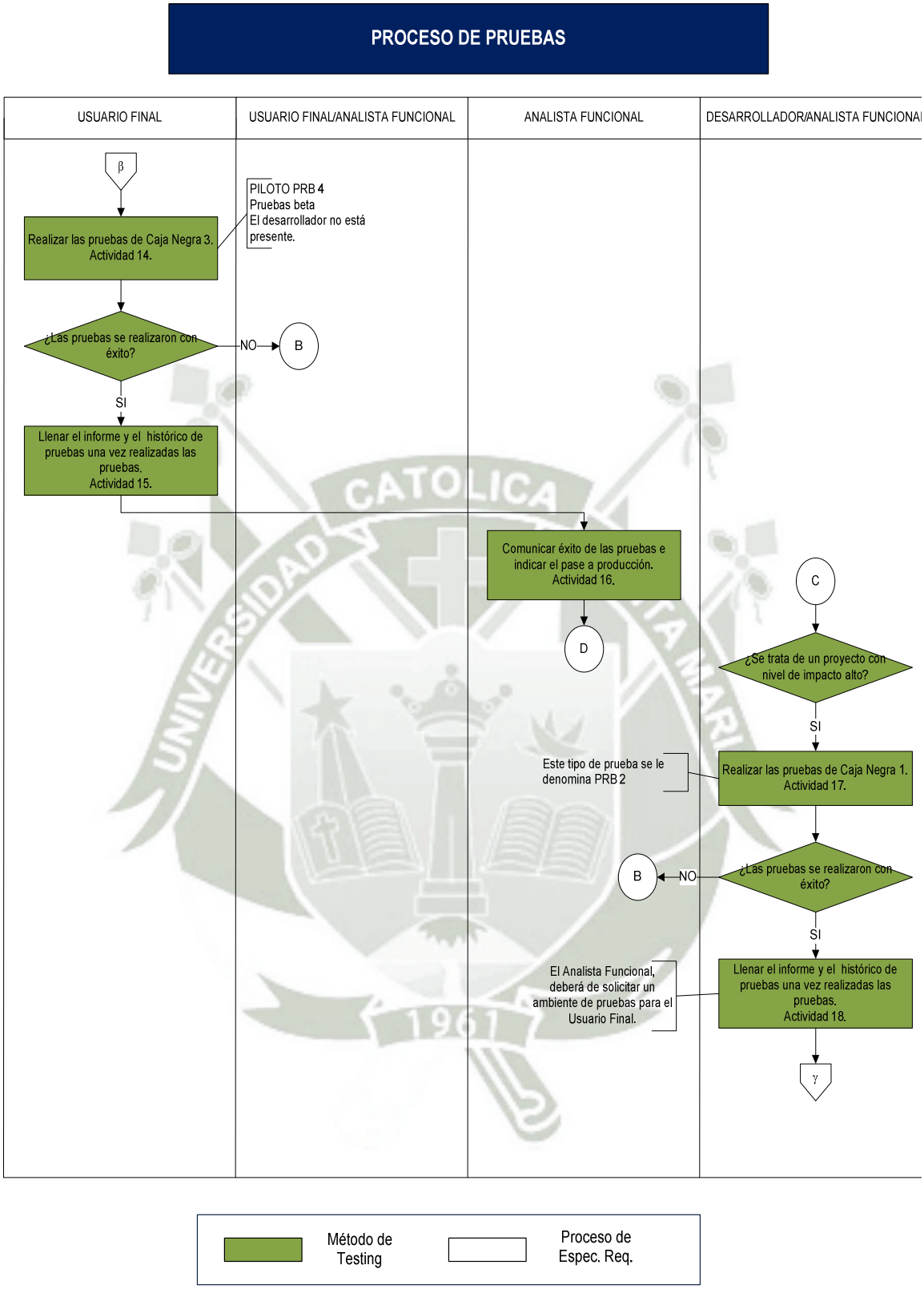

Figura 3.9: Continuación del Proceso de Pruebas del Desarrollador con el Analista Funcional y el Usuario Final.

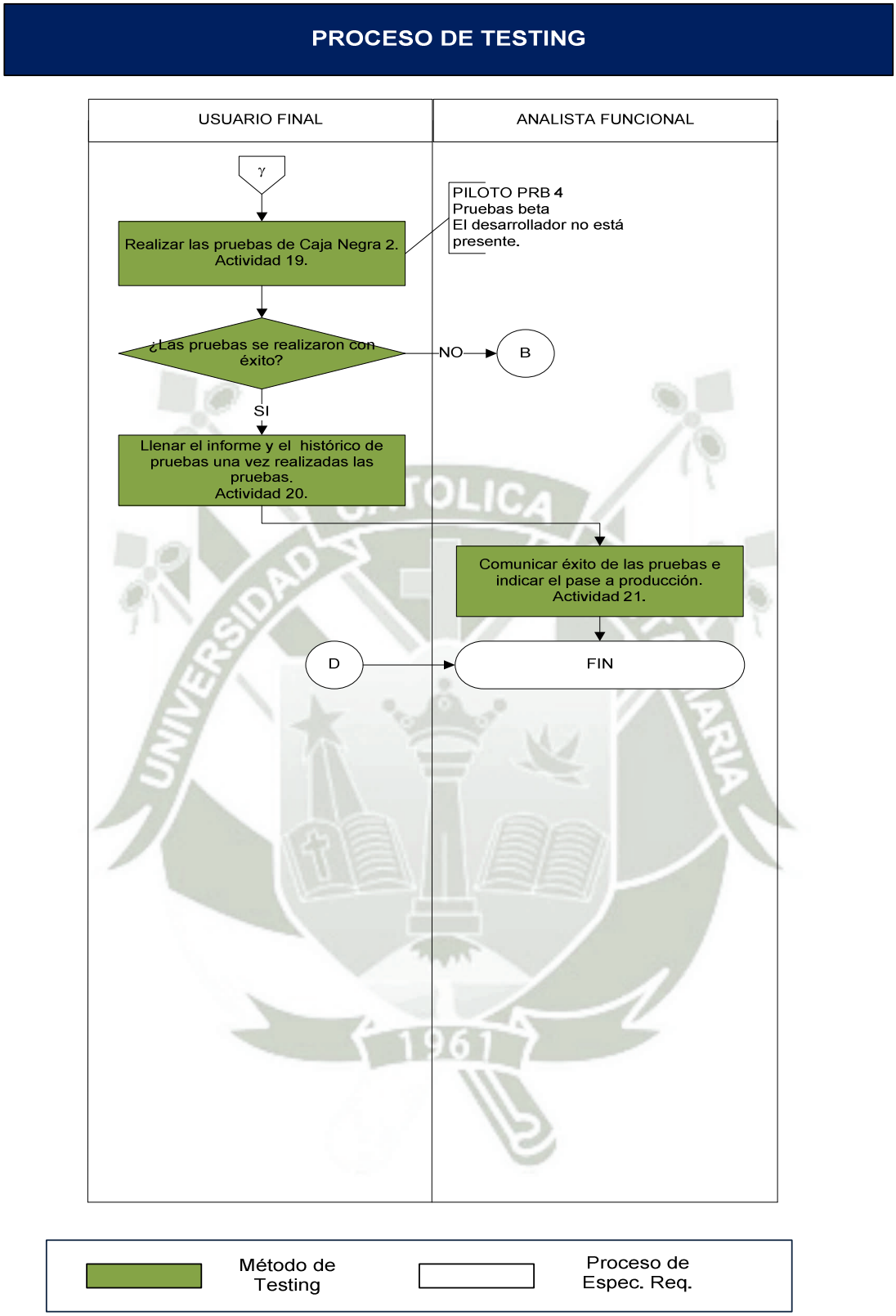

Figura 3.10: Continuación del Proceso de Pruebas del Desarrollador con el Analista Funcional y el Usuario Final.

3.6.1. Testing para Caja Negra:

El Analista Funcional, propone el nivel de impacto que tiene el requerimiento a presentarse; el Comité de Aprobación decide finalmente si el nivel de impacto propuesto es correcto o no. Una vez decidido, se realizara el test case de acuerdo al nivel de impacto. Se clasifica de la siguiente manera:

a) *Nivel de Impacto Alto:*

- Caja Negra 1; el Analista Funcional, elabora el test case con validaciones del Usuario Final; cuando el Desarrollador indica que está listo para las pruebas de Caja Negra, el Analista Funcional realiza junto con el Desarrollador las pruebas elaboradas, esta fase de pruebas se le etiqueta como **PRB 2 (Desarrollador con Analista Funcional).** También se le conoce como pruebas alfa, en donde el Desarrollador actúa como observador.
- Caja Negra 2; una vez concluidas exitosamente las pruebas de Caja Negra 1 o PRB 2, el Analista Funcional junto al Usuario Final, ejecutan las pruebas de Caja Negra 2 o **PRB 3 (Analista Funcional con Usuario Final**);estas pruebas se desarrollan en un ambiente de pruebas preparado por el desarrollador.
- Caja Negra 3; La última etapa para esta categoría, se denomina Caja Negra 3 o **PRB 4(Pruebas Piloto);** en esta fase, el Usuario Final, realiza pruebas en un sistema piloto, donde él sin necesidad de haber elaborado un test case, ejecuta las pruebas que él cree conveniente para evaluar las validaciones solicitadas, también conocido como pruebas beta, en donde el desarrollador no está presente.

b) *Nivel de Impacto Bajo:*

- Caja Negra 1; se denomina pruebas de Caja Negra 1, a las pruebas realizadas por el **Desarrollador con el Analista Funcional (PRB 2),** aquí se realizan pruebas

alfa, y se ejecuta el test case realizado por el Analista Funcional de acuerdo a la validación del Usuario Final.

- Caja Negra 2; esta fase de prueba se le etiqueta como **PRB 4 (Pruebas Piloto);** el Usuario Final, ejecuta sus propias pruebas en el sistema piloto implementado para pruebas(pruebas beta).

3.7. Formatos a seguir

3.7.1. Requerimientos de Software

Se conoce como requerimiento de software, a la necesidad documentada sobre la funcionalidad de la necesidad del usuario en el sistema, estos identifican las características que el sistema debe de cumplir para satisfacer la necesidad.

En el siguiente formato, se presenta los campos necesarios para poder cumplir con la especificación de la necesidad, bajo el estándar IEE 830 – Especificación de Requisitos de Software (ERS).

El requerimiento está compuesto por los siguientes campos:

INTRODUCCIÓN

- **Propósito:** Describe cual es el propósito del documento y detallará a quién va dirigido el requerimiento.
- **Situación actual:** Describe como se encuentra el sistema actualmente, y la necesidad a satisfacer con la implementación del requerimiento.
- **Definiciones, Acrónimos y Abreviaturas**: Se definirán aquí todos los términos, acrónimos y abreviaturas utilizadas en el desarrollo de la ERS.
- **Referencias:** Se presentará una lista completa de todos los documentos referenciados en la ERS.

DEFINICIÓN

- **Perspectiva del Producto:** Se describe la funcionalidad a implementar, se toma en cuenta el producto y sus características descritas de manera general.
- **Funciones del Producto:** Esta subsección debe proporcionar un resumen de las funciones principales que el software debe llevar a cabo. Se pueden utilizar métodos textuales o gráficos, siempre que dichos gráficos reflejen las relaciones entre funciones y no el diseño del sistema.
- **Restricciones:** Se debe indicar aquí cualquier tipo de limitación como pueden ser políticas de la empresa, limitaciones en hardware, seguridad, protocolos de comunicación, interfaces con otras aplicaciones, estándares de la empresa en cuanto a interfaces, etc.
- **Suposiciones y dependencias:** Describe cualquier factor, que si cambia puede afectar a los requerimientos. Si cambian ciertos detalles puede ser necesario revisar los requisitos.

REQUISITOS ESPECÍFICOS

- **Interfaces Externas:** Se definen los requisitos que afecten a la interfaz de usuario.
- **Requisitos Funcionales:** Especifica todas aquellas acciones o funciones que deberá llevar a cabo el sistema a desarrollar.
- **Requisitos No Funcionales:** Se incluyen los requisitos relacionados con los que se espera que tenga que soportar el sistema.

ANEXOS

Se incluirá cualquier tipo de información relacionada con la ERS, pero que no forme parte de la misma.

FORMATO:

<LOGO>	REQUERIMIENTO FUNCIONAL	ERS – 001	
	ESTANDAR IEEE 830: ESPECIFICACION DE REQUISITOS DE SOFTWARE (ERS)	Versión:	1.0
		Fecha:	14/03/2014
		Diseñado:	0001
		Aprobado:	0666
Usuario:			
Fecha:			

INTRODUCCIÓN	
Propósito	
Situación Actual	
Definiciones, acrónicos, abreviaturas.	
Referencias	
DEFINICIÓN	
Perspectiva del Producto	
Funciones del Producto	
Restricciones	
Suposiciones y Dependencias	
REQUISITOS ESPECÍFICOS	
Interfaces Externas	
Requisitos Funcionales	
Requisitos No Funcionales	
ANEXOS	

3.7.2. Viabilidad

Viabilidad del Requerimiento, refiere a las probabilidades de llevarse a cabo o de concretarse de acuerdo a las circunstancias o características del propio requerimiento presentado.

El siguiente formato, muestra los datos relevantes para presentar la viabilidad del requerimiento:

- *Viabilidad:* Indica, si el requerimiento presentado es viable o no de acuerdo a las características presentadas.
- *Prioridad:* Muestra el nivel prioritario (o de nivel impacto); puede ser alta, media o baja.
- *Recurso(s) Asignado(s):* Se presenta la cantidad de recursos estimados para elaborar la implementación del requerimiento.
- *Tiempo estimado:* Propone el tiempo que podría demorar en implementar adecuadamente el requerimiento presentado.
- *Responsable:* Indica quien será el responsable de la ejecución del requerimiento.
- *Comentarios:* Se llenan los comentarios luego de la aprobación del comité.

<LOGO>	**VIABILIDAD DEL REQUERIMIENTO**	**VIA-001**	
		Versión:	1.0
		Fecha:	14/03/2014
		Diseñado:	001
		Aprobado:	066
ID DE REQ.			
FECHA			
Viabilidad			
Prioridad			
Recurso(s) asignado(s)			
Tiempo estimado			
Responsable			
Comentarios			

3.7.3. Documentación de Pruebas

La documentación de esta propuesta de método de testing está basada en el estándar IEEE 829: DOCUMENTACION DE PRUEBAS DE SOFTWARE

- *PLAN DE PRUEBAS*

1. Identificador único del documento
2. Introducción y resumen de elementos y características a probar: Describe el alcance, la aproximación, los recursos y la planificación y las actividades necesarias. Identifica elementos de prueba, las características que deben probarse, las tareas de prueba y lo que hará cada tarea.
3. Elementos: Software que se van a probar (por ejemplo, programas o módulos).
4. Características que se van a probar: Fluidez de datos, independencia de módulos, soporte del software, interfaz de usuario.
5. Criterios de paso/fallo para cada elemento.
6. Criterios de suspensión y requisitos de reanudación.
7. Necesidades de entorno.
8. Necesidades de personal y de formación.
9. Riesgos asumidos por el plan y planes de contingencias para cada riesgo.
10. Aprobaciones y firmas con nombre y puesto desempeñado.

<LOGO>	**PLAN DE PRUEBAS**	**PS – 001**	
	ESTANDAR IEEE 829: DOCUMENTACIÓN DE PRUEBAS DE SOFTWARE	Versión:	1.0
		Fecha:	15/03/2014
		Diseñado:	001
		Aprob.:	0666

ID del documento	*Introducción y resumen de elementos y características a probar*	*Elementos*	*Criterios de paso/fallo*	*Criterios de suspensión y requisitos de reanudación*	*Necesidad del entorno*	*Responsabilidades en la organización y realización de las pruebas.*

Necesidades de personal y de formación	
Riesgos asumidos por el plan y planes de contingencias para cada riesgo	
Aprobaciones y firmas con nombre y puesto desempeñado	

- *ESPECIFICACIÓN DEL DISEÑO DE PRUEBAS*

1. Identificador (único) para la especificación.
2. Características a probar de los elementos software (y combinaciones de características).

3. Detalles sobre el plan de pruebas del que surge este diseño, incluyendo las técnicas de prueba específica y los métodos de análisis de resultados.
4. Identificación de cada prueba: Identificador, casos que se van a utilizar, procedimientos que se van a seguir.
5. Criterios de paso/fallo de la prueba (criterios para determinar si una característica o combinación de características ha pasado con éxito la prueba o no).

<LOGO>	**DISEÑO DE PRUEBAS**	**DS – 001**	
	ESTANDAR IEE 829: **DOCUMENTACIÓN DE PRUEBAS DE SOFTWARE**	Versión:	1.0
		Fecha:	15/03/2014
		Diseñado:	0001
		Aprob.:	0666

ID	*Características a probar*	*Detalle sobre el plan de pruebas*	*Identificador de cada prueba*			*Criterios de paso/fallo de la prueba*
			ID	*Casos de Prueba*	*Procedimiento*	

- HISTORICO DE PRUEBAS:

El histórico de pruebas (test log) documenta todos los hechos relevantes ocurridos durante la ejecución de las pruebas

1. Identificador
2. Descripción de la prueba: elementos probados y entorno de la prueba Anotación de datos sobre cada hecho ocurrido (incluido el comienzo y el final de la prueba)
3. Fecha y hora
4. Identificador de informe de incidente
5. Otras informaciones

<LOGO>	**HISTÓRICO DE PRUEBAS**		**HD – 001**	
	ESTANDAR IEEE 829: DOCUMENTACIÓN DE PRUEBAS DE SOFTWARE		Versión:	1.0
			Fecha:	15/03/2014
			Diseñado:	0001
			Aprob.:	0666
ID	*Descripción de la prueba*		*Fecha y Hora*	*Identificador de informe de incidente*	*Otra Información*
	Elementos probados	*Entorno de la prueba*			

- INFORME DE INCIDENTE

El informe de incidente (test incident report) documenta cada incidente (por ejemplo, una interrupción en las pruebas debido a un corte de electricidad, bloqueo del teclado, etc.) ocurrido en la prueba y que requiera una posterior investigación.

1. Identificador
2. Resumen del incidente
3. Descripción de datos objetivos (fecha/hora, entradas, resultados esperados, etc.)
4. Impacto que tendrá sobre las pruebas

<LOGO>	**INFORME DE INCIDENTE**		**INI – 001**	
	ESTANDAR IEEE 829: DOCUMENTACIÓN DE PRUEBAS DE SOFTWARE		Versión:	1.0
			Fecha:	15/03/2014
			Diseñado:	0001
			Aprob.:	0666

ID	*Resumen de Incidente*	*Descripción de datos objetivos*			*Impacto sobre las pruebas*
		Fecha y Hora	*Entradas*	*Resultados esperados*	

- *INFORME RESUMEN DE PRUEBAS*

El informe resumen (test summary report) resume los resultados de las actividades de prueba (las señaladas en el propio informe) y aporta una evaluación del software basada en dichos resultados

1. Identificador
2. Resumen de la evaluación de los elementos probados

3. Variaciones del software respecto a su especificación de diseño, así como las variaciones en las pruebas
4. Resumen de los resultados obtenidos en las pruebas
5. Evaluación de cada elemento software sometido a prueba (evaluación general del software incluyendo las limitaciones del mismo)
6. Firmas y aprobaciones de quienes deban supervisar el informe

<LOGO>	**INFORME DE RESUMEN DE PRUEBAS**		**INREP - 001**	
	ESTANDAR IEEE 829: DOCUMENTACIÓN DE PRUEBAS DE SOFTWARE		Versión:	1.0
			Fecha:	15/03/2014
			Diseñado:	0001
			Aprob.:	0666
ID	*Resumen de la evaluación de los elementos aprobados*	*Variaciones del software respecto a su especificación de diseño*	*Resumen de los resultados obtenidos*	*Evaluación de cada elemento software sometido a prueba*
Firmas y Aprobaciones				

CAPÍTULO 4

IMPLEMENTACIÓN DEL MÉTODO PARA REALIZAR PRUEBAS DE SOFTWARE

4.1. Proceso del Desarrollo de Pruebas.

El siguiente diagrama muestra de forma resumida, el procedimiento que sigue el desarrollo de las pruebas desde la entrega del requerimiento hasta el pase a producción; e indica el momento se documenta cada caso de prueba.

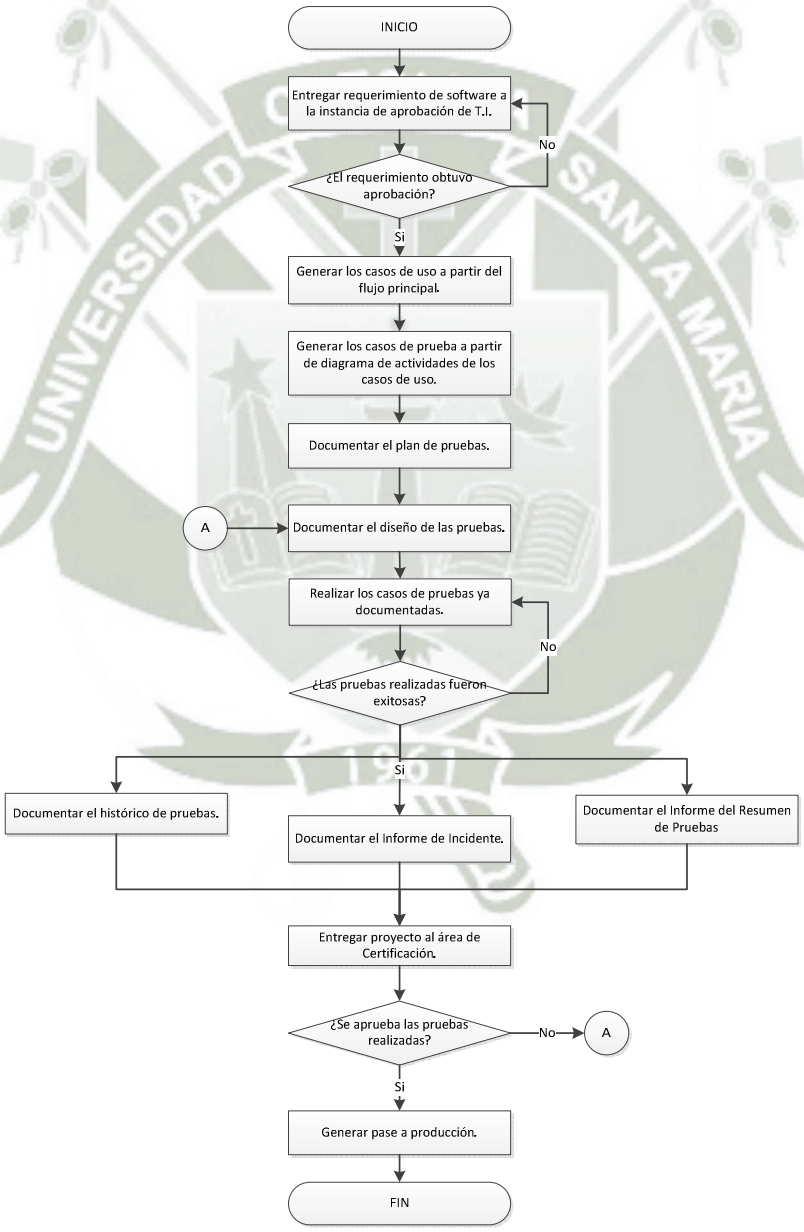

Diagrama 4.1: Proceso de prueba de parte del Analista Funcional.

4.2. Entrega de Requerimiento

Se documenta en "Especificación de Requisitos de Software (ERS)" el Requerimiento Funcional; este requerimiento trata sobre la necesidad de obtener la carga familiar del cliente al momento de solicitar un crédito para una mejor evaluación del mismo.

<table>
<tr><td rowspan="5"><LOGO></td><td colspan="2">ESPECIFICACION DE REQUISITOS DE SOFTWARE (ERS) REQUERIMIENTO FUNCIONAL</td><td colspan="2">ERS - 001</td></tr>
<tr><td rowspan="4"></td><td>Versión:</td><td>1.0</td></tr>
<tr><td>Fecha:</td><td>14/03/2014</td></tr>
<tr><td>Diseñado:</td><td>0001</td></tr>
<tr><td>Aprobado:</td><td>0666</td></tr>
<tr><td>Usuario:</td><td colspan="3">3919</td></tr>
<tr><td>Fecha:</td><td colspan="3">16/03/2014</td></tr>
<tr><td colspan="5"></td></tr>
<tr><td colspan="5">INTRODUCCIÓN</td></tr>
<tr><td>Propósito</td><td colspan="4">Conocer la capacidad de pago de acuerdo a la carga familiar ingresada del cliente.</td></tr>
<tr><td>Situación Actual</td><td colspan="4">Actualmente, no se tiene contemplado los datos de la carga familiar del cliente.</td></tr>
<tr><td>Definiciones, acrónicos, abreviaturas.</td><td colspan="4">• Capacidad de pago: Constituye el principio fundamental de la evaluación de deudores, la cual se determina sobre la base del análisis financiero, la capacidad de generación de flujos de caja provenientes de las actividades propias del giro del negocio, su estabilidad, su tendencia, la suficiencia de los mismos en relación con la estructura de pasivos del deudor ajustados al ciclo productivo del negocio y los factores internos y externos que podrían motivar una variación de la capacidad de pago tanto en el corto como en el largo plazo. Lo cual refleja la posibilidad de que un Prestatario actual o potencial para honrar sus Obligaciones y mantener en el</td></tr>
</table>

<table>
<tr><td></td><td>tiempo un nivel de Solvencia.
• Carga Familiar: Se conoce como carga familiar, a la cantidad de hijos dependientes que cuenta el futuro cliente de la entidad.</td></tr>
<tr><td>Referencias</td><td>No aplica</td></tr>
<tr><td colspan="2">DEFINICIÓN</td></tr>
<tr><td>Perspectiva del Producto</td><td>Se implementará el control sobre la cantidad de hijos de acuerdo al tipo de ingresos de la persona (tipo de persona) el cual revisará si puede acceder a un tipo de producto, y de acuerdo a ese análisis se debe de revisar el monto propuesto a solicitar.</td></tr>
<tr><td>Funciones del Producto</td><td>Gestión de datos de familia del cliente.</td></tr>
<tr><td>Restricciones</td><td>No se contempla la edad ni nivel de instrucción de los hijos.</td></tr>
<tr><td>Suposiciones y Dependencias</td><td>El sistema toma en cuenta dentro de la capacidad de pago del cliente, el gasto asignado en la asignación familiar ingresado en su evaluación económica.</td></tr>
<tr><td colspan="2">REQUISITOS ESPECÍFICOS</td></tr>
<tr><td>Interfaces Externas</td><td>No Aplica</td></tr>
<tr><td>Requisitos Funcionales</td><td>
<table>
<tr><td>RF - 1</td><td>Capturar la información ingresada en el campo Carga Familiar.</td></tr>
<tr><td>Actores:</td><td>Auxiliar de créditos</td></tr>
<tr><td>Descripción:</td><td>Revisar que se haya ingresado un dato en el campo Carga Familiar.</td></tr>
<tr><td>Secuencia del Requerimiento</td><td>El sistema deberá revisar que el usuario haya ingresado información en el campo de Carga Familiar.
En caso, no se haya ingresado ningún dato, el sistema debe de mostrar un mensaje de alerta para que el usuario ingrese información.</td></tr>
<tr><td>RF - 2</td><td>Agregar el valor de “Carga Familiar” como control al</td></tr>
</table>
</td></tr>
</table>

		seleccionar un tipo de producto.
	Actores:	Auxiliar de créditos
	Descripción:	Controlar que al seleccionar el producto, se verifique la carga familiar.
	Secuencia del Requerimiento	El sistema deberá revisar la información ingresada en el campo de Carga Familiar. Y comparar si está de acuerdo a lo permitido al momento de seleccionar el producto a proponer la solicitud del Analista. En caso, la carga familiar sobre pase a lo permitido de acuerdo al producto, el sistema debe de mostrar un mensaje restrictivo solicitando autorización
	RF - 3	Revisar la validación del valor de Carga Familiar de acuerdo al producto y verificar el monto propuesto
	Actores:	Auxiliar de créditos
	Descripción:	Controlar que al ingresar el monto propuesto, se revise el producto de acuerdo a la carga familiar.
	Secuencia del Requerimiento	El sistema deberá revisar el monto propuesto con el producto seleccionado y validar los campos de Carga Familiar, para no sobrepasar el límite establecido. En caso, la carga familiar sobre pase a lo permitido del monto con respecto al producto, el sistema debe de mostrar un mensaje restrictivo solicitando autorización.
	RF – 4	Agregar solicitud de autorización para excepciones de producto por carga familiar.
	Actores:	Auxiliar de créditos
	Descripción:	Generar autorizaciones al Administrador para excepciones de producto por carga familiar.
	Secuencia del	El sistema debe de realizar el procedimiento de

Requerimiento	autorizaciones.
RF – 5	Agregar solicitud de autorización para excepciones de monto por carga familiar.
Actores:	Auxiliar de créditos
Descripción:	Generar autorizaciones al Administrador para excepciones de monto por carga familiar.
Secuencia del Requerimiento	El sistema debe de realizar el procedimiento de autorizaciones.
RF - 6	Modificar la impresión de documentos contractuales
Actores:	Asesor de Servicios
Descripción:	Imprimir los documentos contractuales con el valor de carga familiar.
Secuencia del Requerimiento	El sistema debe de grabar el valor de Carga Familiar, y al momento de realizar la impresión de documentos contractuales, el sistema debe de mostrar impreso el valor grabado.
RF - 1	Generar reporte de excepciones de productor por Carga Familiar.
Actores:	Gerente Territorial
Descripción:	Generar un reporte mostrando los clientes los cuales hayan sido aprobados con excepciones de producto por Carga Familiar.
Secuencia del Requerimiento	El sistema deberá generar un reporte diario mostrando todos los créditos que hayan sido aprobados con excepciones de producto por Carga Familiar.
RF - 1	Generar reporte de excepciones de monto por Carga Familiar.
Actores:	Gerente Territorial

	Descripción:	Generar un reporte mostrando los clientes los cuales hayan sido aprobados con excepciones de monto por Carga Familiar.
	Secuencia del Requerimiento	El sistema deberá generar un reporte diario mostrando todos los créditos que hayan sido aprobados con excepciones de monto por Carga Familiar.
	RF - 1	Generar reporte de Carga Familiar.
	Actores:	Gerente Territorial
	Descripción:	Generar un reporte mostrando los clientes que registren carga familiar mayor a cero.
	Secuencia del Requerimiento	El sistema deberá generar un reporte diario mostrando todos los créditos que hayan sido aprobados sin excepciones de Carga Familiar.
Requisitos No Funcionales	**Rendimiento:**	El servidor de Base de Datos deberá tener un respaldo apropiado para soportar que los usuarios ingresen al sistema de manera simultánea.
	Seguridad:	Uso de contraseñas.
	Disponibilidad:	El sistema debe estar disponible en todo momento.
	Comunicación:	El sistema manejará mensajes de error como también de confirmaciones exitosas.
	Portabilidad:	Utilizar el lenguaje y plataforma actual.
ANEXOS	No Aplica	

4.3. Evaluación de la Viabilidad del Requerimiento

Se documenta, la aprobación del comité de T.I. donde se indica la viabilidad del requerimiento a iniciar.

<LOGO>	VIABILIDAD DEL REQUERIMIENTO	VIA-001	
		Versión:	1.0
		Fecha:	14/03/2014
		Diseñado:	001
		Aprobado:	066

ID DE REQ.	ERS – 001
FECHA	16/03/2014
Viabilidad	Sí
Prioridad	Media
Recurso(s) asignado(s)	1 Persona a cargo
Tiempo estimado	2 semanas
Responsable	Coordinador de desarrollo
Comentarios	La modificación permite la mejor evaluación y registro del cliente.

4.4. Generación de Casos de Prueba

4.4.1. Creación de Casos de Uso

A continuación; se crean los casos de uso para identificar el actor que interactuará con el sistema, así también se identifica la secuencia que realiza el actor y se especifica la funcionalidad del caso.

Nombre:	UC – 01 Modificar en la base de datos el valor a Carga Familiar
Actores:	Auxiliar de créditos
Pre Condición:	No Aplica
Secuencia principal:	1. El usuario ingresa los datos principales del cliente 2. El usuario ingresa el dato en el campo “Carga Familiar”. 3. El usuario graba la operación. 4. El sistema revisa el dato ingresado sea dentro del límite establecido en la base de datos
Secuencia alternativa:	Si el campo “Carga Familiar” se encuentra vacío, el sistema muestra un mensaje de error y solicita se ingrese un valor.
Post Condición:	El dato ingresado quede grabado en el sistema.

Nombre:	UC – 02 Mostrar el dato de la variable “nCarFam”
Actores:	No Aplica
Pre Condición:	Variable creada en la base de datos
Secuencia principal:	1. El sistema muestra los datos ingresados de Carga Familiar.
Secuencia alternativa:	No Aplica
Post Condición:	El dato ingresado quede grabado en el sistema.

Nombre:	UC – 03 Añadir valor a Carga Familiar
Actores:	Auxiliar de créditos
Pre Condición:	No Aplica
Secuencia principal:	1. El usuario ingresa los datos principales del cliente 2. El usuario ingresa el dato en el campo “Carga Familiar”. 3. El usuario graba la operación.
Secuencia alternativa:	Si el campo “Carga Familiar” se encuentra vacío, el sistema muestra un mensaje de error y solicita se ingrese un valor.
Post Condición:	El dato ingresado quede grabado en el sistema.

Nombre:	UC – 04 Controlar el valor de Carga Familiar con el producto a ofrecer
Actores:	Auxiliar de créditos
Pre Condición:	El usuario ingresó un valor en el campo Carga Familiar.
Secuencia principal:	1. El usuario ingresa a la solicitud del cliente. 2. El sistema muestra el formulario de la solicitud. 3. El usuario ingresa los datos de la solicitud del cliente. 4. El usuario graba la operación. 5. El sistema revisa la carga familiar ingresada.
Secuencia alternativa:	Si el producto seleccionado no permite la cantidad de carga familiar, debe de mostrar un mensaje de autorización.
Post Condición:	Validación del producto con la carga familiar.

Nombre:	UC – 05 Controlar el valor de Carga Familiar con el producto a ofrecer y el monto propuesto
Actores:	Auxiliar de créditos
Pre Condición:	El usuario ingresó un valor en el campo Carga Familiar.
Secuencia principal:	1. El usuario ingresa a la solicitud del cliente. 2. El sistema muestra el formulario de la solicitud. 3. El usuario ingresa los datos de la solicitud del cliente. 4. El usuario graba la operación. 5. El sistema revisa la carga familiar ingresada.
Secuencia alternativa:	Si el monto propuesto sobrepasa el límite permitido dentro del producto seleccionado y no permite la cantidad de carga familiar, debe de mostrar un mensaje de autorización.
Post Condición:	Validación del monto de acuerdo al producto con la carga familiar.

Nombre:	UC – 06 Solicitar autorización por producto de excepción por carga familiar.
Actores:	Instancia de Aprobación

Pre Condición:	El número ingresado en carga familiar excede el permitido por producto
Secuencia principal:	1. El usuario 1 envía solicitud de autorización al usuario 2. 2. El sistema registra solicitud de autorización y la coloca en la lista de solicitud del usuario 2. 3. El usuario 2 ingresa a las solicitudes de autorización. 4. El sistema muestra la solicitud registrada del usuario 1. 5. El usuario autoriza la excepción. 6. El sistema graba la autorización. 7. El usuario 1ingresa nuevamente a la solicitud del cliente. 8. El sistema muestra la autorización aprobada. 9. EL usuario 1 graba la solicitud de excepción.
Secuencia alternativa:	Si la autorización fue denegada se rechaza automáticamente la solicitud del crédito.
Post Condición:	Estado de la solicitud grabada o rechazada.

Nombre:	UC – 07 Solicitar autorización por monto de excepción por carga familiar.
Actores:	Instancia de Autorización
Pre Condición:	El número ingresado en carga familiar excede el permitido por monto
Secuencia principal:	1. El usuario 1 envía solicitud de autorización al usuario 2. 2. El sistema registra solicitud de autorización y la coloca en la lista de solicitud del usuario 2. 3. El usuario 2 ingresa a las solicitudes de autorización. 4. El sistema muestra la solicitud registrada del usuario 1. 5. El usuario autoriza la excepción. 6. El sistema graba la autorización. 7. El usuario 1ingresa nuevamente a la solicitud del cliente. 8. El sistema muestra la autorización aprobada.

	9. EL usuario 1 graba la solicitud de excepción.
Secuencia alternativa:	Si la autorización fue denegada se rechaza automáticamente la solicitud del crédito.
Post Condición:	Estado de la solicitud grabada o rechazada.

Nombre:	UC – 08 Modificar documentos contractuales
Actores:	Asesor de Servicios
Pre Condición:	Registros en el sistema el valor ingresado en Carga Familiar.
Secuencia principal:	1. El usuario ingresa al módulo de Servicios. 2. El usuario ingresa el código del cliente. 3. El sistema muestra la solicitud aprobada del cliente ingresado. 4. El usuario presiona “Generar Cronograma”. 5. El sistema genera el cronograma de pagos del cliente ingresado. 6. El usuario graba la operación. 7. El sistema genera los documentos contractuales con los datos del cliente, incluyendo el ingreso de Carga Familiar. 8. EL usuario imprime los documentos contractuales generados.
Secuencia alternativa:	No Aplica.
Post Condición:	No Aplica.

Nombre:	UC – 09 Generar el reporte de excepción de producto de carga familiar.
Actores:	Territorial
Pre Condición:	Registros en el sistema de excepciones de producto a carga familiar.
Secuencia principal:	1. El usuario ingresa al módulo de Reportes. 2. El sistema muestra las opciones de Reportes que puede revisar. 3. El usuario selecciona la opción de Reportes a excepciones de carga

	familiar. 4. El sistema muestra el reporte generado.
Secuencia alternativa:	Si no hubiese datos registrados como excepciones se muestra un mensaje indicando que no se encuentran datos.
Post Condición:	No Aplica.

Nombre:	UC – 10 Generar el reporte de excepción de monto de carga familiar.
Actores:	Territorial
Pre Condición:	Registros en el sistema de excepciones de monto a carga familiar.
Secuencia principal:	1. El usuario ingresa al módulo de Reportes. 2. El sistema muestra las opciones de Reportes que puede revisar. 3. El usuario selecciona la opción de Reportes a excepciones de carga familiar. 4. El sistema muestra el reporte generado.
Secuencia alternativa:	Si no hubiese datos registrados como excepciones se muestra un mensaje indicando que no se encuentran datos.
Post Condición:	No Aplica.

Nombre:	UC – 11 Generar el reporte de carga familiar.
Actores:	Territorial
Pre Condición:	Registros en el sistema de carga familiar.
Secuencia principal:	1. El usuario ingresa al módulo de Reportes. 2. El sistema muestra las opciones de Reportes que puede revisar. 3. El usuario selecciona la opción de Reportes de carga familiar. 4. El sistema muestra el reporte generado.
Secuencia alternativa:	No Aplica
Post Condición:	No Aplica.

4.4.2. Generación de Diagrama de Actividades a Partir de Casos de Uso

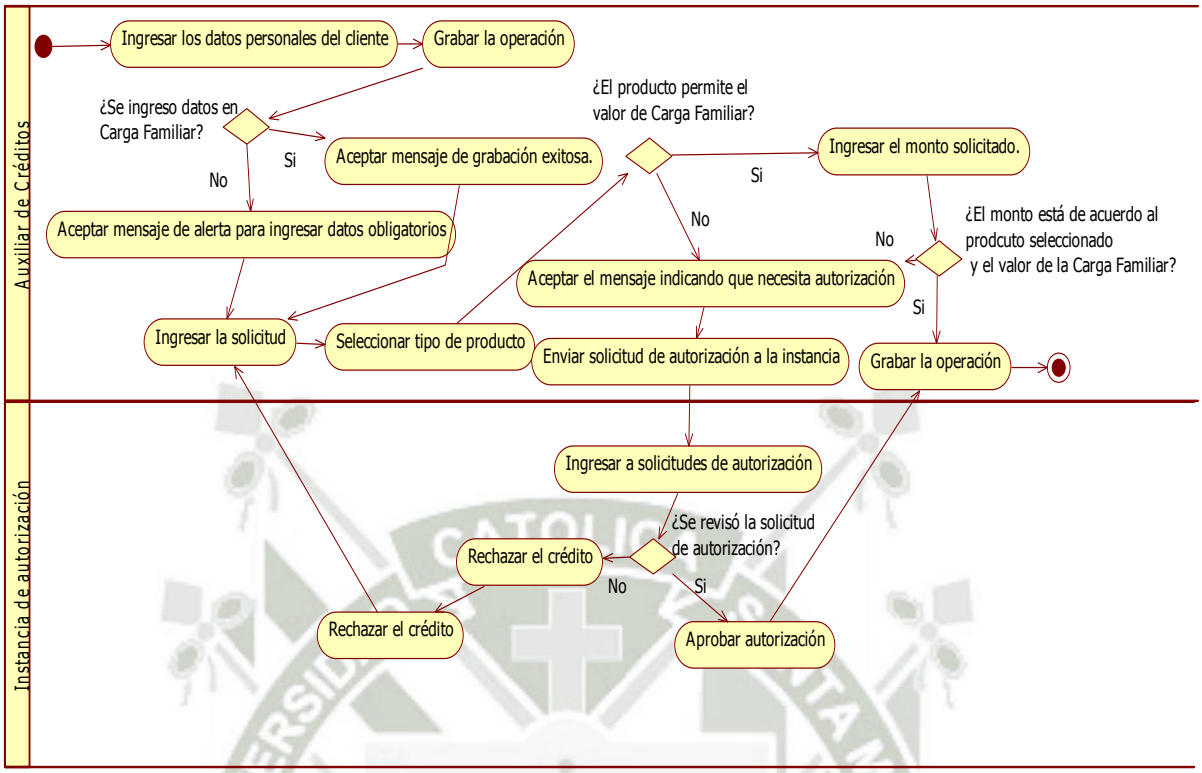

Figura 4.2: Diagrama de Actividades para la implementación del valor de Carga Familiar.

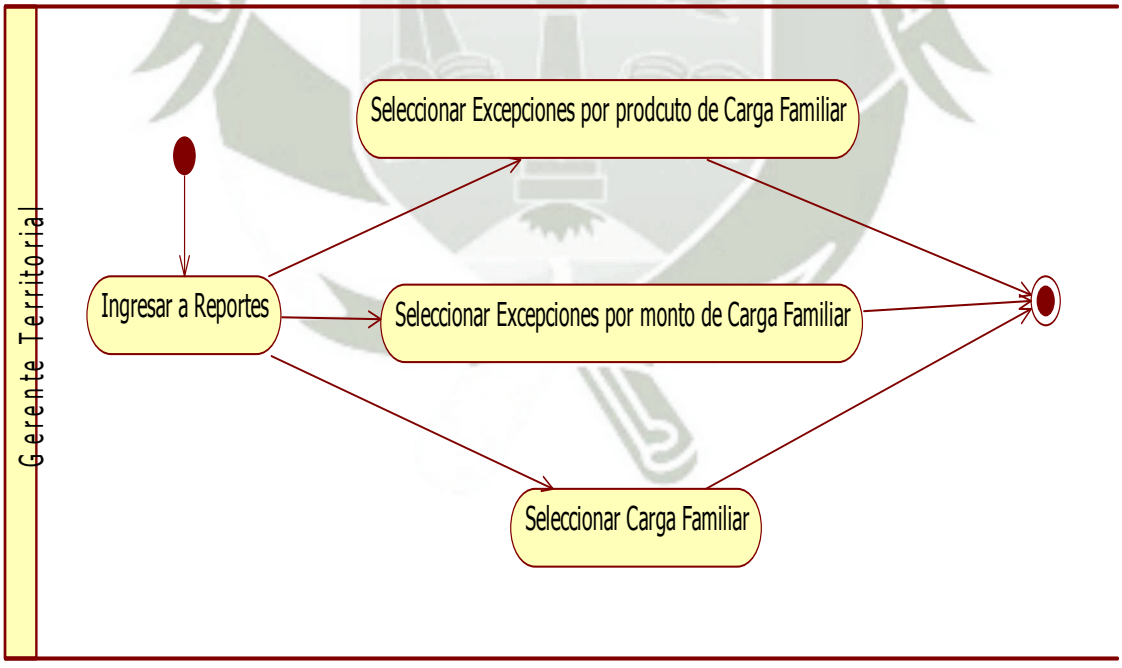

Figura 4.3: Diagrama de Actividades para la implementación de Reporte de Carga Familiar

TECNICAS DE CAJA NEGRA

a) Particiones de Equivalencia

Condición	Clases Válidas	Clases Inválidas
Número de Hijos	1. Número de dos dígitos mayor igual a ‘00’.y menor igual a ‘99’.	2. Número menos de 2 dígitos. 3. Número mayor de 2 dígitos. 4. Número negativo. 5. No es número. 6. Cadena Nula.

b) Valores Límite

Tabla 1

Valores Límite para el caso de prueba

Condición	Clases Válidas	Clases Inválidas
Número de hijos	1. Número ‘02’	2. Número ‘2’. 3. Número ‘002’. 4. ‘-5’ 5. ‘A’ 6. -

Fuente Propia

4.5. Documentación de Casos de Prueba

<LOGO>	PLAN DE PRUEBAS DOCUMENTACIÓN DE PRUEBAS DE SOFTWARE	PS – 001	
		Versión:	1.0
		Fecha:	15/03/2014
		Diseñado :	001
		Aprob.:	0666

ID	*Introducción y resumen de elementos y características a probar*	*Elementos*	*Criterios de paso/fallo*	*Criterios de suspensión y requisitos de reanudación*	*Necesidad del entorno*	*Responsabilidades en la organización y realización de las pruebas.*
CP_1	Ingresar datos en el campo de Carga Familiar.	Módulo de Créditos	Paso: No ingresar datos y muestra mensaje de error. Fallo: Ingresa dato y no permite guardar.	No Aplica	Sistema financiero	Pruebas de Documentación: usuario 3919 Pruebas de software: usuario 3919
CP_2	Controlar el producto de acuerdo a la Carga Familiar.	Módulo de Créditos	Paso: Revisar el tipo de producto. Fallo: Sobrepase el límite permitido	No Aplica	Sistema financiero	Pruebas de Documentación: usuario 3919 Pruebas de software: usuario 3919
CP_3	Controlar el monto de acuerdo al producto según Carga Familiar.	Módulos de Créditos	Paso: Revisar el monto. Fallo: Sobrepase el límite permitido	No Aplica	Sistema financiero	Pruebas de Documentación: usuario 3919 Pruebas de software: usuario 3919

CP_4	Solicitar autorización para excepción por producto	Módulo de Créditos	Paso: Solicitud Enviada Fallo: No se registre solicitud	No Aplica	Sistema financiero	Pruebas de Documentación: usuario 3919 Pruebas de software: usuario 3919
CP_5	Solicitar autorización para excepción por monto	Módulo de Créditos	Paso: Solicitud Enviada Fallo: No se registre solicitud	No Aplica	Sistema financiera	Pruebas de Documentación: usuario 3919 Pruebas de software: usuario 3919
CP_6	Modificar documentos contractuales	Módulo de Créditos	Paso: Mostrar en documentos el valor de carga familiar Fallo: No mostrar ningún dato de carga familiar	No Aplica	Sistema financiera	Pruebas de Documentación: usuario 3919 Pruebas de software: usuario 3919
CP_7	Generar reportes	Módulo de Créditos	Paso: Generar reporte Fallo: Mostrar datos válidos	No Aplica	Sistema financiera	Pruebas de Documentación: usuario 3919 Pruebas de software: usuario 3919
Necesidades de personal y de formación			Permisos para ingresar al módulo específico			
Riesgos asumidos por el plan y planes de contingencias para cada riesgo			Copias de seguridad Frecuencia Periodicidad Plan de contingencias Prever fallos críticos Procedimientos alternativos Tratamiento de errores Posibilidad de error recuperación Planificación contenido mensajes de error			
Aprobaciones y firmas con nombre y puesto desempeñado						

<LOGO>	DISEÑO DE PRUEBAS DOCUMENTACIÓN DE PRUEBAS DE SOFTWARE	DS – 001	
		Versión:	1.0
		Fecha:	15/03/2014
		Diseñado:	0001
		Aprob.:	0666

ID	*Características a probar*	*Detalle sobre el plan de pruebas*	*Identificador de cada prueba*			*Criterios de paso/fallo de la prueba*
			ID	*Casos de Prueba*	*Procedimiento*	
DS – 003	Número de Hijos	CP - 1	UC – 03	Añadir valor a Carga Familiar	Ingresar el valor al campo Carga Familiar	Paso: Grabar el valor permitido Fallo: Grabar un valor no permitido
DS – 004	Producto a ofrecer	CP – 2	UC – 04	Controlar el valor de Carga Familiar con el producto a ofrecer	Seleccionar el producto luego de haber ingresado el dato de Carga Familiar y grabar.	Paso: Grabar con autorización en caso sobre pase lo permitido. Fallo: No grabar.
DS – 005	Monto propuesto	CP – 3	UC – 05	Controlar el valor de Carga Familiar con el producto a ofrecer y el monto propuesto	Ingresar el monto luego de haber ingresado el dato de Carga Familiar y grabar.	Paso: Grabar con autorización en caso sobre pase lo permitido. Fallo: No grabar.
DS	Solicitar	CP - 4	UC – 06	Solicitar	Sobre pasar	Paso:

– 006	autorización de excepción por producto			autorización por producto de excepción por carga familiar.	el límite permitido y grabar.	Enviar autorización Fallo: Grabar de manera normal
DS-007	Solicitar autorización de excepción por monto.	CP – 5	UC – 07	Solicitar autorización por monto de excepción por carga familiar.	Sobre pasar el límite permitido y grabar.	Paso: Enviar autorización Fallo: Grabar de manera normal
DS-008	Impresión de documentos	CP – 6	UC – 08	Modificar documentos contractuales	Realizar la impresión de documentos.	Paso: Visualizar el dato de Carga Familiar Fallo: No visualizar el dato de Carga Familiar
DS-009	Generación de reportes	CP - 7	UC – 09	Generar el reporte de carga familiar.	Ingresar al módulo de Reportes y seleccionar los reportes a visualizar	Paso: Mostrar datos correctos incluyendo excepciones Fallo: Ausencia de datos y de excepciones

CAPÍTULO 5

EVALUACIÓN DEL MÉTODO PARA REALIZAR PRUEBAS DE SOFTWARE

5.1. Evaluación por Expertos

Se conoce como evaluación por expertos, aquella evaluación que se realiza a un grupo de personas las cuales presentan un perfil adecuado para poder dar opinión acerca de lo presentado basado en su experiencia.

Dado que la tesis está enfocada a las medias empresas financieras de la ciudad de Arequipa y éstas son 4; Compartamos Financiera, Caja Municipal de Arequipa, Financiera Nueva Visión y Caja Rural Incasur, en cada una de ellas se solicitará entrevistar a la persona encargada de las pruebas de software, más dos personas más con experiencia suficiente en pruebas de software e informática financiera, es que se estima tener, como máximo, seis personas expertas en estos menesteres para poder tener opiniones importantes y, consecuentemente, resultados significativos.

5.2. Perfil del Experto:

a) Profesión: Ingeniería de Sistemas

b) Experiencia: 2 años en banca, sistemas financieros

c) Cantidad de evaluadores: 6

5.3. Cuestionario

Cuestionario acerca de la propuesta de Metodología de Testing:

1. ¿Cree usted que la metodología es fácil de aplicar?

Es factible ☐

Es complejo ☐

2. ¿Cree usted que la metodología es eficaz al proceso del ciclo de vida del software?

Bueno ☐

Regular ☐

Malo ☐

3. ¿Cree usted que la metodología está clara para el personal al momento de realizar pruebas?

Muy Bueno ☐

Bueno ☐

Regular ☐

Malo ☐

4. ¿Cree usted que la metodología es atractiva para el usuario?

Si ☐

No ☐

5. ¿Considera que la metodología propuesta para realizar pruebas, es capaz de ejecutarse en cualquier entidad?

Si es competente ☐

Es inepto ☐

6. En su opinión sobre la metodología; ¿usted cree que es útil para la entidad?

Muy útil ☐

Útil ☐

Inútil ☐

7. ¿Se podría decir que la metodología propuesta es eficaz para disminuir errores antes de llegar a la puesta en producción del requerimiento solicitado?

Muy eficaz ☐

Eficaz ☐

Ineficaz ☐ 8. ¿Cree usted que la metodología ofrece ventaja para ofrecer una mejor calidad de software al usuario? Muy ventajosa ☐ Ventajosa ☐ Desventajosa ☐

5.4. Respuestas de Cuestionario

Luego de aplicado el cuestionario al grupo de expertos de una entidad financiera, se obtuvieron los siguientes resultados:

Tabla 2:

Respuesta a la Pregunta 1:

¿Cree usted que la metodología es fácil de aplicar?							**Total**
Es factible	1		1	1	1		**4**
Es complejo		1				1	**2**
Total							**6**

Fuente: Elaboración propia

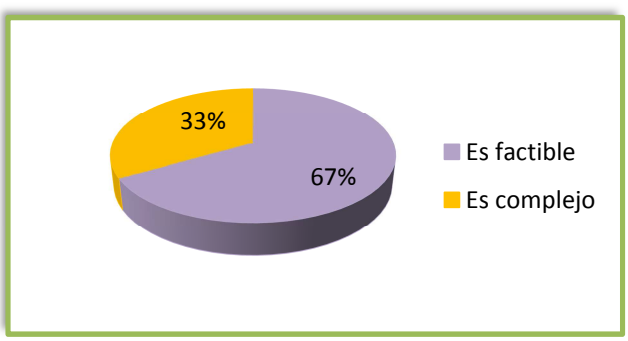

Figura 5.1: Muestra de resultado de la pregunta 1

El 67% indica que la metodología propuesta es factible para poder ser aplicada en la institución financiera.

Tabla 3:

Respuesta a la Pregunta 2:

¿Cree usted que la metodología es eficaz al proceso del ciclo de vida del software?							**Total**
Bueno							**0**
Regular	1		1	1	1	1	**5**
Malo		1					**1**
Total							**6**

Fuente: Elaboración propia

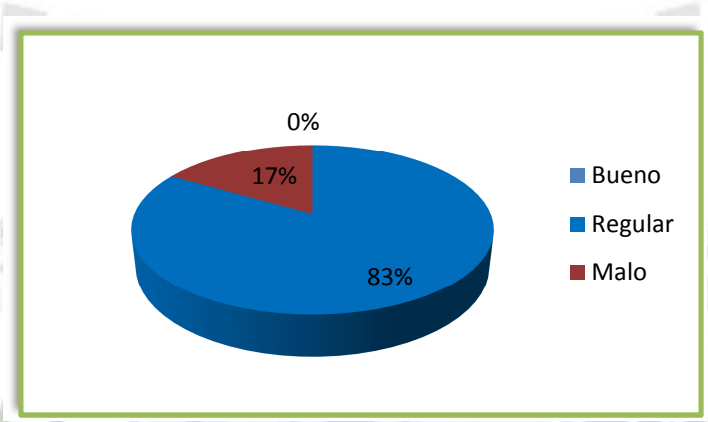

Figura 5.2: Muestra de resultado de la pregunta 2

De acuerdo al cuestionario el 83% considera que la metodología propuesta es eficaz de manera regular en el ciclo de vida del software.

Tabla 4:

Respuesta a la Pregunta 3:

¿Cree usted que la metodología está clara para el personal al momento de realizar pruebas?							**Total**
Muy Bueno							**0**
Bueno	1		1	1			**3**
Regular		1			1		**2**
Malo						1	**1**
Total							**6**

Fuente: Elaboración propia

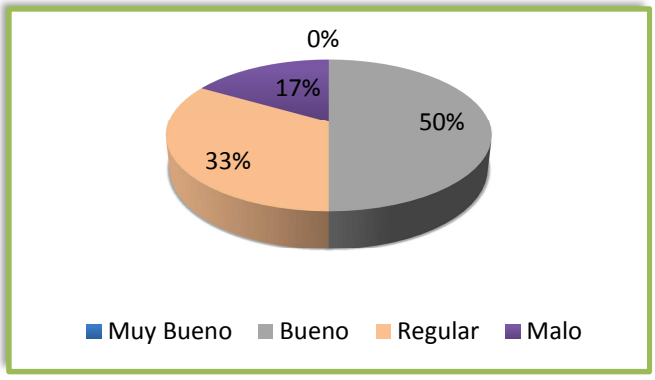

Figura 5.3: Muestra de resultado de la pregunta 3

La mitad de la selección de expertos (50%), indica según el cuestionario que la metodología propuesta se expresa de manera clara en un índice bueno para el personal que desarrolla pruebas de software en la entidad.

Tabla 5:

Respuesta a la Pregunta 4:

¿Cree usted que la metodología es atractiva para el usuario?							**Total**
Si				1	1		**2**
No	1	1	1			1	**4**
Total							**6**

Fuente: Elaboración propia

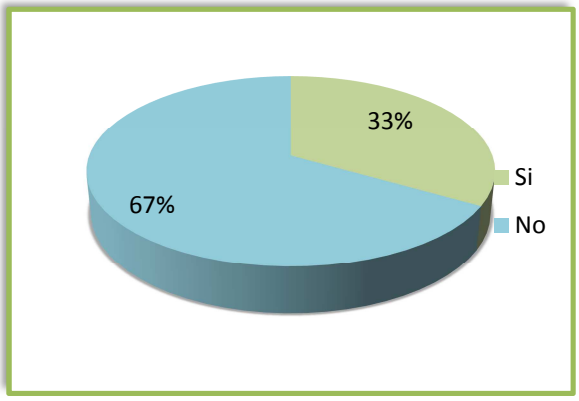

Figura 5.4: Muestra de resultado de la pregunta 4

Dentro del grupo de expertos, el 67% de entrevistados, muestran la negativa como respuesta a la pregunta si la metodología propuesta es atractiva a los usuarios, dado que, para el personal con poca experiencia sería complicado aprender aplicarla.

Tabla 6:

Respuesta a la Pregunta 5:

¿Considera que la metodología propuesta para realizar pruebas, es capaz de ejecutarse en cualquier entidad?							**Total**
Si es competente	1		1	1		1	**4**
Es inepto		1			1		**2**
Total							**6**

Fuente: Elaboración Propia

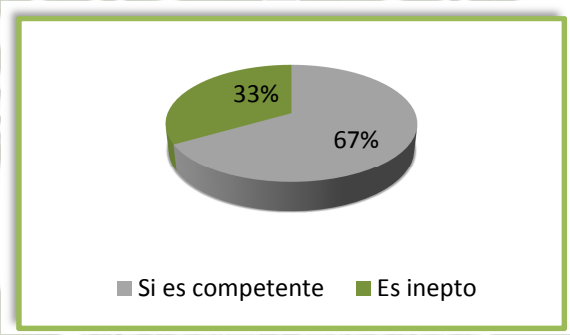

Figura 5.5: Muestra de resultado de la pregunta 5

El 67% de los entrevistados, señalan que la metodología propuesta para realizar pruebas, tiene la capacidad de ser ejecutada en cualquier entidad

Tabla 7:

Respuesta a la Pregunta 6:

En su opinión sobre la metodología; ¿usted cree que es útil para la entidad?							**Total**
Muy útil							**0**
Útil	1		1	1	1		**4**
Inútil		1				1	**2**
Total							**6**

Fuente: Elaboración propia

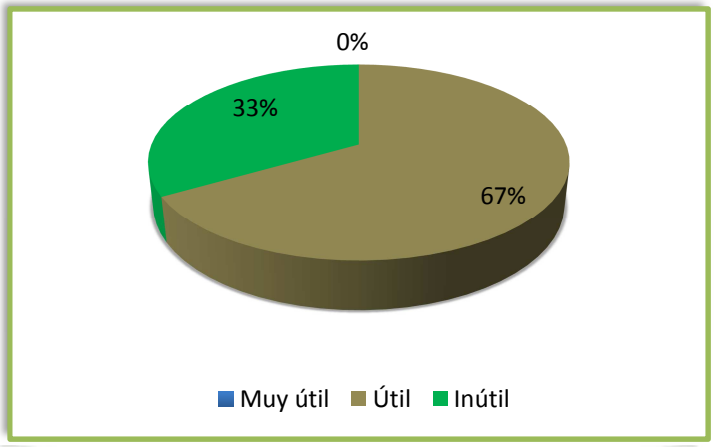

Figura 5.6: Muestra de resultado de la pregunta 6

El 67% del grupo seleccionado de expertos, indican que la metodología propuesta es útil para la entidad financiera.

Tabla 8:

Respuesta a la Pregunta 7:

¿Se podría decir que la metodología propuesta es eficaz para disminuir errores antes de llegar a la puesta en producción del requerimiento solicitado?							**Total**
Muy eficaz							**0**
Eficaz	1		1	1	1		**4**
Ineficaz		1				1	**2**
Total							**6**

Fuente Elaboración propia

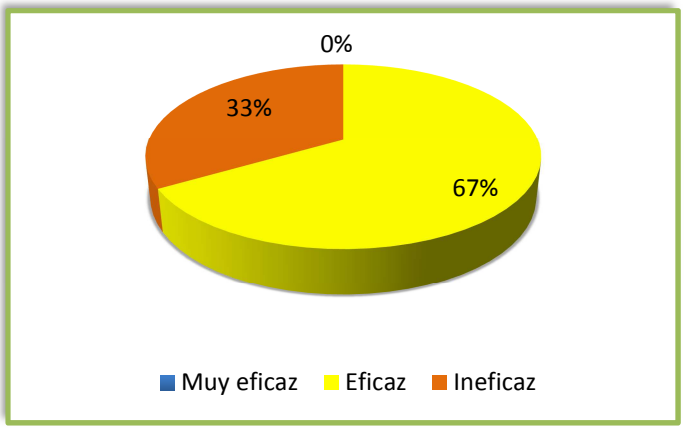

Figura 5.7: Muestra de resultado de la pregunta 7

El 67% del grupo de expertos, coinciden que la metodología propuesta es eficaz para disminuir errores antes de llegar a la puesta en producción.

Tabla 9:

Respuesta a la Pregunta 8:

¿Cree usted que la metodología ofrece ventaja para ofrecer una mejor calidad de software al usuario?							**Total**
Muy ventajosa							**0**
Ventajosa	1		1		1	1	**4**
Desventajosa		1		1			**2**
Total							**6**

Fuente Elaboración Propia

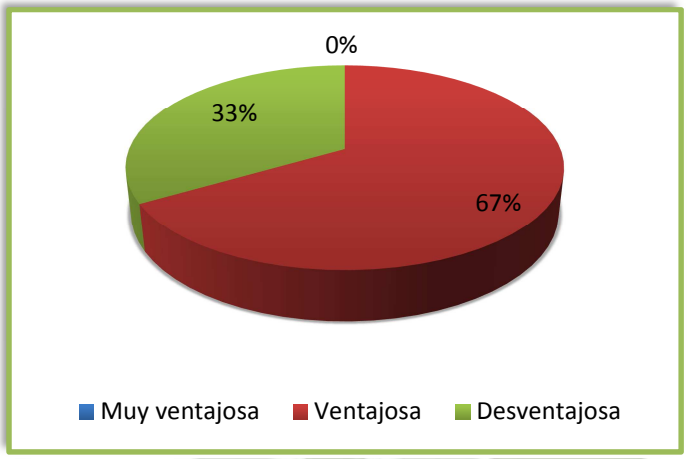

Figura 5.8: Muestra de resultado de la pregunta 8

El 67% del grupo de expertos, consideran que la metodología propuesta es ventajosa para ofrecer un software con mayor calidad.

CONCLUSIONES

1. Utilizando las diferentes técnicas de caja negra para realizar pruebas de validación de software, se deduce que es factible implementar un método con el mismo fin, derivando los casos de prueba de los casos de uso, se asegura que se hayan cumplido los requisitos funcionales del sistema.
2. Generando un plan de pruebas, en base a los diferentes diagramas, ayuda a determinar la menor cantidad de errores en el sistema luego de implementar el requisito funcional solicitado.
3. Con este método, se llega a una coordinación entre el desarrollo y la prueba, esta ventaja se aprovecha cuando el personal encargado de realizar las pruebas tiene la experiencia necesaria para poder comprender el método y toda su funcionalidad.

RECOMENDACIONES

1. Desarrollar un software que dé soporte al método propuesto para gestionar las pruebas, sacar estadísticas e ir mejorándolo de acuerdo a la retro-alimentación que se puede obtener.
2. Ampliar el método para que cubra las pruebas de las instituciones financieras de todo tamaño en el Perú.
3. Proponer el método para que las organizaciones de supervisión del Perú lo tomen como referencia en sus auditorías.
4. Generalizar el método para que pueda servir como un estándar de auditoría de pruebas de software en organizaciones de mediano tamaño.

BIBLIOGRAFIA

1. Gutiérrez Javier J., Escalona María J., Mejías Manuel y Reina Antonia M., Modelos de Pruebas para Pruebas del Sistema, Universidad de Sevilla, 2006
2. Gutiérrez Javier Jesús, Escalona María José, Mejías Manuel, Torres Jesús, Estudio Comparativo de Propuestas para la Generación de Casos de Prueba a Partir de Requisitos Funcionales, Universidad de Sevilla, 2005
3. It- Mentor, Pruebas de software, "Capacitación y Guía para el Desarrollo de Software"
4. Pérez Lamancha Beatriz, Proceso de Testing Funcional Independiente, Montevideo Uruguay 2006,
5. Pérez Lamancha Beatriz, Metodología del Trabajo, Laboratorio Testing Funcional, Montevideo Uruguay 2006
6. Pressman Roger S. Ingeniería de Software, un enfoque práctico. Madrid: Mc Graw Hill. 2005
7. Sommerville Ian. Ingeniería del Software. Madrid: Pearson Educación S.A. 2005

TESIS

8. Infantas Luque Tatiana Sofía, Desarrollo de un Modelo que Permita la Gestión Cuantitativa de Procesos de Mantenimiento de Software , 2005
9. Mamani Amanqui, Flor Karina, Modelo de Auditoria Informática Aplicado a la Especificación de Requerimientos de Software de Sistemas de Información, 2008
10. Orihuela Pastor Alvaro Antonio, Metodología para la Especificación de Requisitos de un Producto de Software de Calidad Bajo ISO 9000, 2000

11. Rivera Guillen, Briggi, Análisis y Optimización del Procesos de Pruebas del Software para una Empresa de Seguros, 2012.
12. Vásquez Lerzundi Wilfredo, Modelo de Elicitacion de Requerimientos en Sistemas de Software Construidos Mediante el Proceso Unificado, 2008