Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas

“Estrategia de pruebas para líneas de desarrollo de componentes de software en la UCI”

Autores: Elizabeth de la Paz Gómez

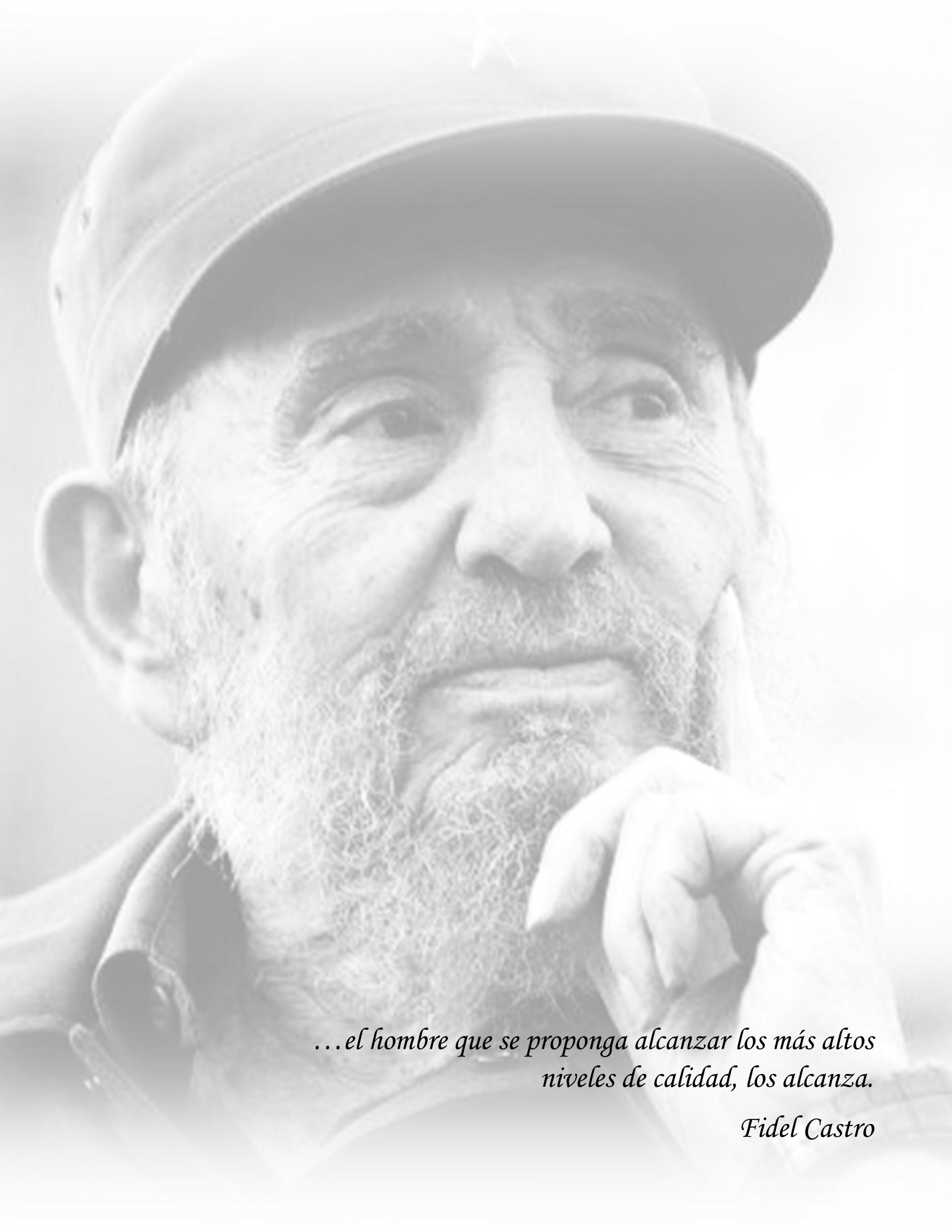
Aryanna Echevarría Fiallo

Tutores: MSc.Reynaldo Álvarez Luna

Ing. Dairys Febles Pérez

La Habana, junio de 2015

“Año 57 de la Revolución”



…el hombre que se proponga alcanzar los más altos niveles de calidad, los alcanza.

Fidel Castro

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los ___ días del mes de _____ del año _____.

Elizabeth De La Paz Gómez

_____

Firma de Autor

Aryanna Echevarría Fiallo

_____

Firma de Autor

MSc.Reynaldo Álvarez Luna

_____

Firma de Tutor

Ing. Dairys Febles Pérez

_____

Firma de Tutor

Tutores

1. MSc.Reynaldo Álvarez Luna

   Correo: rluna@uci.cu

   Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

2. Ing. Dairys Febles Pérez

   Correo: dfebles@uci.cu

   Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

Autores

1. Elizabeth de la Paz Gómez

   Correo: edelapaz@estudiantes.uci.cu

   Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

2. Aryanna Echevarría Fiallo

   Correo: aechevarria@estudiantes.uci.cu

   Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

Agradecimientos

A mis padres por su amor y cariño, por la educación que me han dado, por ser ejemplos de consagración, sacrificio y por ser la calidad de revolucionarios que son.

A la luz de mi vida, Manuel Ernesto, por su amor y dedicación, por todos los momentos que hemos compartido, por todas las veces que me ha levantado el ánimo y me ha hecho seguir adelante, por ayudarme a ser una mejor persona, por haberme ayudado a hacer realidad este sueño, por construir la familia tan linda que hoy tenemos.

A mis tutores Dairys y Reynaldo, porque a pesar de estar contra el tiempo han sabido guiarnos en cada momento de la investigación.

A mis hermanos Eyisel y Alien, por todos los momentos compartidos.

A Manolo y Belquita por haberse convertido en más que mis suegros, mi familia y por haberme dado cada día su amor, su cariño y por preocuparse tanto por mí.

A Aryanna, mi dúo de tesis por hacer realidad este sueño junto a mí.

A yure y el negro por ser más que mis amigos unos hermanos para mí, por el apoyo incondicional que me han dado siempre, por todos los bellos momentos que hemos compartido.

A Dayron y Piro, por ser mis hermanitos de todos los tiempos.

A Yailin e Irela muchas gracias por convertirse en mis amigas incondicionales y por todo el apoyo que me han brindado.

A Liuver, Orlando, la rubia, el suri, Eddy y Yasel porque ocupan un lugarcito en mí.

A Google por tener siempre una respuesta para mí.

Gracias a todos los que de una forma u otra hicieron que este sueño sea hoy una realidad.

De Elizabeth.

Agradecimientos

Es difícil agradecer a todas las personas que de una forma u otra hicieron este sueño realidad. Fueron largos años de estudio, conocí a personas que nunca voy a olvidar, de buenos y malos momentos estuvo conformado mis 7 años en la universidad. Cuando me fui de la universidad nunca pensé que tendría otra oportunidad, pues convertirme en trabajadora 2 años y volver a ser estudiante son dos momentos muy diferentes a los cuales tuve que enfrentar.

Comenzar agradeciendo a mi papá el cual me ha dado su apoyo incondicional en todo momento.
A mi mamita que la amo con todo mi corazón, gracias por cuidarme tanto a pesar de que me haz malcriado.
A mis hermanos queridos, darle las gracias por ser un ejemplo a seguir y por quererme tanto.
No puedo dejar de mencionar a dos personitas que brillan incluso en mis días más oscuros. David y Miguel Antonio, mis sobrinos que por ser tan pequeños no saben que tienen una tía que da la vida por ellos.
A mi familia, a mis tías Martica, Margarita y Hilda, mis abuelitas María y Carito, a mis primos, a todos en general ya que han aportado su granito de arena para que saliera adelante.
No puedo dejar de mencionar a unas personas que han sido mis otros padres que en los momentos difíciles han dado el paso al frente sin reparar, Ana Rosa y Giraldo.
A Elizabet, mi dúo de tesis que sin ella no hubiera cumplido este sueño.
A Manuel el novio de Elizabet que gastó días y noches de sueño con nosotras sin importar que también tenía que graduarse.
A mis tutores Reynaldo y Dairys, que nos han guiado en todo momento, por su tiempo y entrega.

Agradecimientos

A mis amigas y amigos de la UCI que mencionarlos todos y dejar uno afuera sería imperdonable, pero no pueden faltan mis 4 hermanitas Adrialis, Lianet, Rosaida y Martha ellas fueron lo mejor que me pasó aquí, las tuve en las buenas y en las malas y nunca me fallaron, decirles gracia es poco pues no encuentro la palabra adecuada para describirlas, simplemente son mis otras hermanas.

A Yanet por ser una hermana en estos años, por saber escucharme y cada vez que me enfermaba correr conmigo para el hospital.

A Jeem por ser una persona especial de buenos y malos momentos, por tratar de enseñarme lo que estaba bien y lo que estaba mal.

A mis amigos del grupo que supieron acogerme a pesar de solamente estar con ellos 2 años.

A las lokillas de mi apartamento Laritza, Sonia, Yailin, Glennis y Jessie por compartir conmigo cada momento y brindarme su apoyo.

A Luis Miguel, porque sin su ayuda y la de su laptop hubiera pasado muchísimo trabajo.

A Yoandy mi cuñado y sus amigos por desestresarme cuando más lo necesitaba.

A mis amigos de trabajo "I.L.V Mantenimiento Fabril" que todos y cada uno de ellos dieron lo mejor de sí para que yo fuera una mejor persona.

A mis profesores de años, pues ya que son muchos los que he tenido no los puedo mencionar a todos, simplemente "GRACIAS".

De Aryanna

A mami, que aunque hoy no está hoy entre nosotros, siempre está a mi lado.

De Elizabeth

A mi mamá y mi papá que son lo más lindo del mundo, por darme todo su amor y cariño, por sacrificarse por mí y por ayudarme a llegar donde estoy.

A mis hermanos por ser mi guía para poder llegar tan lejos.

A mi familia en general por todo el apoyo y el amor que me han brindado.

A todo aquel que me ayudó a ser una mejor persona cada día, que me enseñaron a no ver los defectos de los demás sino la bondad que cada cual trae en sí.

Fueron años inolvidables los cuales recordaré por siempre, tengo experiencias para contar.

A mi Dios por acompañarme siempre.

De Aryanna

Resumen

La calidad del software es el factor clave para la aceptación del producto informático por parte del cliente y tributa a la competitividad entre las empresas. La presente investigación surge a partir de la necesidad de la Universidad de las Ciencias Informáticas (UCI), de una técnica, procedimiento o estrategia de pruebas que permita medir la calidad de un componente durante su desarrollo. Para ello se propone una estrategia de pruebas, que facilite y guie el proceso de aplicación de pruebas a un componente de software. La misma se encuentra estructurada por seis fases que a su vez proponen actividades de acuerdo a los niveles de pruebas propuestos por Pressman y los métodos de diseño de casos de pruebas existentes. Para su validación se utilizaron técnicas definidas por el método Delphi, con la colaboración de expertos en los temas de gestión y administración de la calidad de software de la Universidad, lo que permitió concluir que a partir del uso de la estrategia se establecen buenas prácticas para el desarrollo basado en componentes, aumenta la calidad y el nivel de reutilización de los componentes desarrollados, disminuyen los problemas en la integración de los productos y se garantiza la medición y mejora de los indicadores de calidad en la línea de componentes.

Palabras Claves: calidad de software, estrategia de pruebas, líneas de componentes, niveles de pruebas.

Abstract

Software quality is the key factor for the acceptance of the software product by the client and it fosters competitiveness among companies. This research arises from the need for the University of Informatics Sciences (UCI), of a technique, procedure or test strategy that allows measuring the quality of a component during development. To accomplish that it is proposed a test strategy, to facilitate and guide the test implementation process of a software component. It is structured in six phases which in turn propose activities according to the test levels proposed by Pressman and the existing design methods of test cases. To validate it, techniques defined by the Delphi method were used, with the collaboration of experts in the fields of management and software quality management at the University, allowing to conclude that from the use of the strategy good practices for component-based development are established, apart from that increases the quality and the level of reuse of components developed, reduces the problems in the integration of products and the measurement and improvement of quality indicators in the components line is guaranteed.

Key Words: software quality, test strategy, component lines, test levels.

Índice

Índice de Ilustraciones

Índice de Tablas

Introducción

En los últimos años la informática y las telecomunicaciones han evolucionado vertiginosamente y con ello han llevado a la sociedad a lo que se conoce como “Era de la Información”. Debido al desarrollo de la informática, esta se encuentra inmersa en casi todos los campos de la actividad humana tales como: la educación, las artes, la industria, las finanzas, el gobierno y la salud.

Existe una creciente preocupación por lograr que el software que se desarrolla sea un producto con calidad. Para ello, se avanza en la definición e implementación de estándares que fijan los atributos de calidad que debe tener un sistema informático para lo cual surgen modelos y metodologías para la evaluación de la calidad.

La calidad del software es una combinación de factores, que varían entre diferentes aplicaciones, y los requerimientos especificados por el cliente. Diversos autores y estándares definen que es la calidad de software, entre estos se encuentran Roger Pressman, la IEEE (Instituto de Ingeniería Eléctrica y Electrónica) y la norma ISO 9126.

Pressman se refiere a la calidad del producto informático como “la concordancia con los requisitos funcionales y de rendimientos explícitamente establecidos, estándares de desarrollo explícitamente documentados y características implícitas que se espera de todo programa informático desarrollado profesionalmente” (Pressman, 2005).

Para garantizar la calidad de un producto durante su proceso de desarrollo se realizan actividades de verificación y validación, con el objetivo de que el producto cumpla con los requerimientos especificados. En el ámbito de la investigación se utilizará la definición elaborada por Pressman, por su enfoque práctico con respecto a los componentes implícitos en el desarrollo de un sistema informático.

Se han desarrollado diversos modelos para evaluar la calidad de un producto informático entre los que se encuentran la norma ISO 9000, el modelo CMMI (Capability Maturity Model) y el modelo EFQM (European Fundation for Quality Management), con el propósito de obtener un producto con calidad.

La Universidad de las Ciencias Informáticas (UCI), es un centro destinado a la formación de profesionales calificados, producción de software y prestación de servicios informáticos. La UCI se encuentra dividida en siete facultades conformadas por centros de desarrollo. En la facultad 6, en el área de producción está ubicado el Centro de Tecnologías de Gestión de Datos (DATEC). En el 2014 el centro DATEC cambió su

estructura para proporcionar la realización de sistemas informáticos bajo un modelo industrial basado en líneas de componentes. El desarrollo de software basado en líneas de componentes permite que el producto se elabore con mayor rapidez, menor costo y más calidad, ofreciendo significativas e importantes mejoras, pues no solo es una forma o método de desarrollo, también es una manera de aportar calidad al producto (CZARNECKI, 2006).

El trabajo con líneas de componentes minimiza la duplicación de los componentes. Al desarrollar un componente de software se le han de aplicar pruebas de forma independiente con el objetivo de evitar errores al integrarlos (CZARNECKI, 2006). Un componente debe poseer atributos entre los que se encuentran: que sea identificable, genérico, reutilizable dinámicamente, independiente de la plataforma y reemplazable, además que se encuentre auto contenido, que sea accedido solo a través de su interfaz, que sus servicios no varíen y que se encuentre bien documentado.

Debido al poco tiempo de existencia de las líneas de componentes en la Universidad y a la escasa experiencia que poseen los desarrolladores, actualmente productos como: el Gestor de recursos de hardware y software (GRHS), el Generador Dinámico de Reportes (GDR) y otros productos desarrollados presentan diversos problemas entre los que se encuentran:

1. Bajo nivel de reutilización debido a la falta de calidad de los componentes desarrollados.
2. Problemas en la integración de productos, pues una vez integrado los componentes a un sistema sin ser estos probados con anterioridad, pueden surgir errores en el mismo.
3. No se realiza adecuadamente el desarrollo basado en componentes, pues los componentes desarrollados son para un sistema específico, no son reutilizables por otro sistema.
4. Al aplicar pruebas de calidad sólo se ejecutan pruebas funcionales y de rendimiento al software con interfaz de usuario, no se realizan todas las pruebas necesarias para medir las características que posee el componente.

Teniendo en cuenta la problemática descrita anteriormente, se define como **problema de la investigación:**

El sistema de gestión de la calidad en la UCI presenta limitaciones para el aseguramiento de la calidad basado en el modelo de líneas de componentes.

Siendo identificado como **objeto de estudio**: La gestión de la calidad de software en líneas de componentes y dentro de dicho objeto de estudio se enmarca el **campo de acción**: Estrategias de pruebas para líneas de componentes de software en la Universidad.

Para dar solución al problema planteado se define como **objetivo general** de la investigación:
Diseñar una estrategia de pruebas para líneas de desarrollo de componentes de software que avale la calidad de los componentes desarrollados en la Universidad.

Las siguientes **preguntas científicas** guiarán el desarrollo de la investigación:

1. ¿Cuáles son los principales procesos que ha de integrar la estrategia de pruebas para su correcto funcionamiento?
2. ¿Cuáles son las pruebas que se pueden aplicar a los componentes desarrollados bajo una línea de desarrollo de componentes?
3. ¿Cómo diseñar pruebas para líneas de desarrollo de componentes de software?
4. ¿Cómo validar el correcto funcionamiento de la estrategia de pruebas para líneas de desarrollo de componentes de software?

Para dar cumplimiento al objetivo planteado se trazaron las siguientes **tareas de la investigación**:

1. Revisión de los principales métodos y técnicas de prueba existentes para seleccionar los que se utilizarán para la implementación de la estrategia de pruebas para líneas de desarrollo de componentes en la Universidad de las Ciencias Informáticas.
2. Caracterización de las líneas de componentes de productos informáticos y sus particularidades en la Universidad de las Ciencias Informáticas para definir las actividades de calidad para la estrategia de pruebas.
3. Identificación de las sub-prácticas de los procesos de Verificación y Validación del Nivel 3 de CMMI Dev 1.3 para su posterior utilización en la estrategia de pruebas.
4. Revisión de las herramientas y tecnologías de pruebas y gestión de calidad para la selección de las que se utilizarán en el desarrollo de la estrategia de pruebas para líneas de desarrollo de componentes.
5. Realizar una revisión bibliográfica del método Delphi para su utilización en la validación de la estrategia de pruebas para líneas de desarrollo de componentes de software.

Métodos de investigación

El proceso de investigación y desarrollo de la estrategia de pruebas para líneas de componentes de software estuvo guiado por los métodos de investigación científica que se presentan a continuación:

Para profundizar en el estudio de lo relacionado con el flujo de trabajo de prueba y crear las condiciones para analizar las características del proceso de pruebas de software que no se pueden observar directamente se utilizaron los siguientes **métodos teóricos:**

Análisis - síntesis: A través del uso de este método se obtuvo el marco teórico de la investigación, lo cual permitió la integración en el proceso de desarrollo de software y la síntesis para establecer el análisis entre las características y las definiciones de las fases que forman la estrategia de pruebas.

Método de modelación: Este método se utilizó para realizar la modelación de la estrategia de pruebas para líneas de desarrollo de componentes de software en la UCI.

Método sistémico: Consiste en estudiar el objeto mediante la determinación de sus componentes o como componente del objeto al que pertenece, así como la relación entre ellos que conforma una realidad como totalidad. Este método fue utilizado para analizar la calidad del proceso de desarrollo de software en la UCI.

Para la obtención y elaboración de datos empíricos relacionados con el proceso de pruebas de software se utilizan los siguientes **métodos empíricos:**

Se realizó una **entrevista** a los directivos de líneas de desarrollo de componentes del centro DATEC para obtener información relacionada con las pruebas de calidad que se aplican actualmente durante el ciclo de desarrollo de software. En conjunto con este método se utilizó el método **análisis documental,** el cual fue utilizado para la identificación, recopilación y transformación de los documentos utilizados para enunciar las teorías que sustentan el estudio de las pruebas de software. Esta combinación permitió obtener el flujo de actividades de la estrategia de pruebas propuesta.

Para la validación de la estrategia de pruebas se aplicó el **método Delphi**, el cual consiste en la utilización sistemática del juicio intuitivo de un grupo de expertos para obtener un consenso de opiniones informadas. Para la aplicación de método se seleccionó un panel que se encuentra formado por especialistas de diferentes áreas de la Universidad. Se aplicó una **encuesta** a los mismos mediante un cuestionario previamente elaborado, para obtener sus opiniones acerca de la estrategia de pruebas. Una vez

completados los cuestionarios se les realizó un procedimiento de **medición** con el objetivo de obtener información numérica acerca de la estrategia, con el fin de llevar a cabo la validación de la misma.

El presente trabajo de diploma se encuentra organizado en tres capítulos

Capítulo 1: Fundamentación teórica para la estrategia de pruebas para líneas de desarrollo de componentes de software

Este capítulo contiene los principales conceptos investigativos asociados al dominio del problema así como técnicas, metodologías, tecnologías y herramientas utilizadas en el desarrollo de la solución.

Capítulo 2: Estrategia de Pruebas para líneas de componentes

En este capítulo se diseña la estrategia de pruebas y se definen las fases, técnicas de pruebas, roles, responsabilidades, y artefactos generados por fases.

Capítulo 3: Validación de la Estrategia de Pruebas

En este capítulo se describe método de pronóstico cualitativo Delphi y los resultados de la aplicación del mismo con el fin de validar la estrategia de pruebas.

Capítulo 1: Fundamentación teórica de la estrategia de pruebas para líneas de desarrollo de componentes de software

En el presente capítulo se explican los conceptos necesarios para el entendimiento de los objetivos de la investigación. Además se realiza el estudio de los diferentes tipos de pruebas, estrategias, métodos y herramientas que se utilizan en la actualidad, para medir la calidad de componentes de software tanto en el ámbito internacional como nacional (UCI).

1.1. Calidad de software

En el proceso de desarrollo de software, la calidad es un factor de vital importancia. En la calidad se centran el correcto funcionamiento, la seguridad y la confiabilidad del producto. La mayoría de los autores que se han pronunciado sobre el tema de la calidad, plantean que la calidad debe ser definida desde el punto de vista del consumidor, confirmando este planteamiento Joseph Jurán, experto en gestión de la calidad, considera que “Calidad es adecuación al uso del cliente”(Jurán 1989). La ISO 9000 define Calidad como “Grado en el que un conjunto de características inherentes cumplen con los requisitos”. Pressman plantea que un producto tiene calidad cuando se cumple el propósito para el cual fue previsto, se satisfacen totalmente las expectativas de los clientes y consumidores finales y cuando es realizado en el tiempo pronosticado y al presupuesto acorde para el mismo. Por otra parte, La Real Academia de la Lengua Española define calidad como “La propiedad o conjunto de propiedades inherentes a una cosa que permiten apreciarla como igual, mejor o peor que las restantes de su especie” (Española, 2015).

Estos criterios relacionados con la calidad de un producto también son aplicados cuando se trata de desarrollo de software. Actualmente debido el aumento de los productores de sistemas informáticos, se ha experimentado un incremento de la competencia entre las empresas desarrolladoras. Estas desean producir sistemas con alta calidad, en el menor tiempo posible y a costos mínimos. Por esta razón la calidad comienza a ser un punto esencial en la creación de productos para garantizar su presencia en el mercado, situando la calidad del software en un punto importante para el desarrollo de aplicaciones. Este proceso se ha convertido en uno de los grandes problemas con los que se han afrontado los fabricantes de productos informáticos. Por este motivo el proceso de calidad es objeto de estudio de especialistas, ingenieros, investigadores y comercializadores de sistemas informáticos, los cuales han encaminado sus esfuerzos a investigar en torno a las siguientes interrogantes:

- ¿Cómo obtener un software con calidad?
- ¿Cómo evaluar la calidad del software?

Según la norma ISO-8042 (ISO 1994): *"la calidad de software está establecida como el conjunto de propiedades y características de un producto o servicio que le confieren la capacidad para satisfacer necesidades expresadas o implícitas."*

El Institute of Engineers Electrician and Electronic (IEEE, 1990), señala que "*la calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario*".

Según Roger S. Pressman, *"la calidad del software es la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente"* (Pressman, 2005).

La definición dada por Pressman es una de las más completas, ya que en su mayoría los problemas con la calidad del software no se encuentran en el software, sino en la forma que las personas interpretan y aplican el concepto de calidad. El objetivo no sólo está en desarrollar un producto que cumpla con los requerimientos especificados, también radica en que el equipo de desarrollo sea capaz de comprender y adicionar los requerimientos que el consumidor desea pero no es capaz de manifestar. Para obtener un producto con calidad en las entidades o empresas se realizan actividades para medir la calidad, que son previamente planificadas, gestionadas y controladas durante el proceso de desarrollo del software.

1.2. Gestión de la calidad de software

La gestión de calidad es el conjunto de normas correspondientes a una organización, vinculadas entre sí (Campos, 2002). A partir de estas normas, la empresa u organización en cuestión podrá administrar de manera organizada la calidad de la misma. La misión siempre está enfocada hacia la mejora continua de la calidad**.** Para precisar el concepto de gestión de la calidad y poder hacer valoraciones sobre su idoneidad, se introducen diversas normas y marcos de trabajo, entre los que se encuentran la ISO 9001 y CMMI.

Los modelos CMMI (Capability Maturity Model Integration) o Integración de modelos de madurez de capacidades son colecciones de buenas prácticas que ayudan a las organizaciones a mejorar sus procesos. Estos modelos son desarrollados por equipos de productos con miembros procedentes de la industria del

gobierno y del Software Engineering Institute (SEI). Se usan para evaluar el nivel de madurez y la capacidad de una compañía en términos de desarrollo informático. Los niveles son utilizados para conocer la madurez de los procesos que se realizan para producir un sistema informático.

En los proyectos productivos de la UCI se emplea la metodología de desarrollo de software AUP-UCI en unión con el modelo CMMI-DEV v 1.3. Actualmente se trabaja en las áreas de procesos del nivel 2. Centros como CESIN, CEDIN y CEYGE se evaluaron en este nivel en el año 2011, evaluación que se pierde con el paso de 3 años, por lo cual se opta porque este año todos los centros productivos de la Universidad certifiquen dicho nivel. En la Universidad se encuentran definidas diversas actividades para la gestión de la calidad de software, entre las que se encuentran:

- Aseguramiento de la calidad del proceso y el producto.
- Revisiones técnicas formales.
- Pruebas de software.
- Mejora de procesos de software.
- Auditorías integrales a los centros.
- Formación continua del personal.
- Sistema de gestión del conocimiento.

En la propuesta de estrategia de pruebas para líneas de desarrollo de componentes, se incluyen actividades correspondientes a las áreas de proceso verificación y validación, pertenecientes al nivel 3 de CMMI.

1.2.1. Sub-prácticas de los procesos de Verificación y Validación

La verificación y la validación son sub-prácticas que se realizan concurrentemente. La validación tiene como objetivo demostrar que un producto o componente de producto cumple con su uso previsto cuando se ubica en el entorno previsto, es decir asegura que se construyó el componente correcto. La verificación no es más que la comprobación de la correspondencia del producto con los requisitos especificados, es decir asegura que se construyó correctamente (CMMI, 2010).

Verificación

Para realizar la verificación de un componente o software se realizan diferentes actividades entre las que se encuentran:

- **Preparación de la verificación:** En este proceso se realizan actividades como la selección de los productos de trabajo, se establece el entorno y los procedimientos y criterios para la verificación del producto.
- **Realizar las revisiones entre pares:** En este proceso se realizan actividades tales como la preparación, realización de las revisiones entre pares y análisis de los resultados.
- **Verificar los productos de trabajo seleccionados:** En este proceso se realizan actividades entre las que se encuentran la realización de las actividades de verificación y análisis de los resultados.

Validación

Para realizar la validación de un componente o software se realizan diferentes actividades entre las que se encuentran:

- **Preparar la validación:** En este proceso se realizan actividades como la selección de los productos a validar, establecimiento del entorno de validación, los procedimientos y criterios de validación.
- **Validar el producto o los componentes de producto:** En este proceso se realizan actividades como la realización de la validación del producto probado y el análisis de los resultados de la validación.

La validación y la verificación son áreas de procesos que se realizan durante el desarrollo de software. A continuación se mencionan actividades pertenecientes a estas áreas de procesos que se evidencian en la creación de estrategias de pruebas:

- Registro de los resultados de cada prueba.
- Verificación del cumplimiento de los criterios de aceptación para cada prueba.
- Selección de las herramientas de pruebas a utilizar.
- Verificación de la correspondencia de la arquitectura del componente que fue definida al inicio del proceso de desarrollo con la obtenida luego de la implementación.
- Aplicación de pruebas de aceptación, rendimiento y seguridad.
- Verificación de la documentación del componente.
- Se analizan los resultados de los registros de las pruebas.

Todas las actividades anteriormente mencionadas tributan a la calidad del software o componente en desarrollo, validando que cumpla con su uso previsto y verificando los requisitos definidos para el mismo.

La realización de estas actividades es una buena práctica a tener en cuenta para el desarrollo de software basado en líneas de componentes.

1.3. Desarrollo de software basado en líneas de componentes

El desarrollo de software basado en líneas de componentes permite reutilizar piezas de código pre elaborado. Permite realizar diversas tareas, brindando beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión (Montilva, 2002).

Un componente de software es una unidad de composición con interfaces contractualmente especificadas y explícitas sólo con dependencias dentro de un contexto. Puede ser desplegado de forma independiente y es sujeto a la composición de terceros (Szyperski, 2002).

Características claves para que un elemento pueda ser catalogado como componente:

- **Identificable**: Debe tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación.
- **Auto contenido:** Un componente no debe requerir de la utilización de otros para concluir la función para la cual fue diseñado.
- **Puede ser reemplazado por otro componente:** Se puede reemplazar por nuevas versiones y por otro componente que lo reemplace y mejore.
- **Con acceso solamente a través de su interfaz:** Debe asegurar que estas no cambiarán a lo largo de su implementación.
- **Sus servicios no varían:** Las funcionalidades ofrecidas en su interfaz no deben variar, pero si puede variar su implementación.
- **Bien documentado:** Un componente debe estar correctamente documentado para facilitar su búsqueda, actualización o integración con otros.
- **Es genérico:** Sus servicios pueden ser utilizados para varias aplicaciones.
- **Reutilizado dinámicamente:** Puede ser cargado en tiempo de ejecución en una aplicación.
- **Independiente de la plataforma:** Puede operar independientemente del hardware, software o sistema operativo con que se cuente.
- **Acceso:** Permite a un componente hacer uso de otros componentes sin conocer anteriormente que protocolo utilizar para esta iteración.

- **Distribución transparente:** Un componente no necesita conocer la ubicación física de otros componentes para interactuar con ellos.

Es importante comprobar estas características durante el proceso de desarrollo del componente, ya que el mismo está sujeto a cambios a lo largo de su implementación y es de primordial importancia que posea los atributos descritos anteriormente. Los componentes en dependencia de sus características y funcionalidades son clasificables, para lo cual se tienen en cuenta criterios bien definidos.

Criterios de clasificación de componentes

Para llevar a cabo la clasificación de un componente se tienen en cuenta determinados criterios, los cuales se describen a continuación:

- **Tamaño:** Puede ser medido por medio de las métricas utilizadas en diseño orientado a objetos, a través de líneas de código (LDC) u Orientadas a función.
- **Complejidad:** Son utilizadas métricas de tamaño para evaluar la complejidad.
- **Frecuencia de re-uso:** Es el número de veces que ha sido utilizado un componente dentro de distintas aplicaciones.
- **Confiabilidad:** Es la probabilidad de fallo en el funcionamiento del componente dentro de cierto escenario operacional (Rojas, 2004).

Partiendo de estos criterios de clasificación los componentes se pueden clasificar en:

- **Framework:** Los frameworks de componentes proporcionan servicios que soportan un modelo de componentes (Montilva, 2002). Estos modelos son patrones que permiten interactuar entre sí de acuerdo al problema que resuelven y permiten la extensibilidad del framework y su funcionalidad.
- **Bussines Component:** Los componentes de negocio son aquellos componentes especializados en prestar servicios enfocados a un dominio en particular (Loos, 2002).

La principal ventaja del desarrollo de software basado en componentes es la reutilización de los mismos, por lo cual los componentes se diseñan y desarrollan con el objetivo de ser reutilizados por otras aplicaciones, con el fin de reducir el tiempo de desarrollo y mejorar la fiabilidad del producto final. La reutilización de componentes suele suceder principalmente a nivel interno dentro de organizaciones, aunque la realización de componentes comerciales ha comenzado a extenderse en los últimos años. De esta forma

surge el término componentes COTS (Commercial-Off-The-Shelf), los cuales presentan las siguientes características (Wesley, 1999):

- Son vendidos o licenciados al público en general.
- Los mantiene y actualiza el propio vendedor, quien conserva los derechos de la propiedad intelectual.
- Su código no puede ser modificado por el usuario.

La tendencia actual es el desarrollo de software basado en líneas de productos (LPS). De acuerdo al SEI (Software Engineer Institute), una línea de productos de software se refiere a un conjunto de sistemas de software que comparten características y que son desarrollados a partir de un conjunto común de bienes núcleo. De la definición anterior es válido aclarar que los bienes núcleo son la base de la línea de productos e incluyen la arquitectura, componentes reutilizables, modelos de dominio, requerimientos, documentación y planes de prueba (Wesley, 1999).

Desde el 2014 en la UCI se trabaja bajo este modelo de producción, combinando el trabajo de líneas de producto con la realización de sistemas basados en componentes. Esta estructura de trabajo es denominada “Desarrollo de software basado en líneas de componentes” (LDCS). El uso de este modelo posee ventajas como:

- **Reutilización del software.** Permite alcanzar un mayor nivel de reutilización de software.
- **Simplifica las pruebas.** Permite que los componentes se prueben por separados, antes de ser ensamblados en el sistema.
- **Simplifica el mantenimiento del sistema.** Cuando existe bajo acoplamiento entre componentes, se presenta la oportunidad de realizar actualizaciones y agregar nuevos componentes sin afectar el sistema.
- **Mayor calidad.** Dado que un componente puede ser construido y luego mejorado continuamente, la calidad de una aplicación basada en componentes puede mejorar con el paso del tiempo.

Todas las ventajas anteriormente expuestas hacen de esta estructura de trabajo una opción viable para el desarrollo de sistemas informáticos. Para su correcta aplicación y con el objetivo de construir sistemas complejos en el menor tiempo posible, es necesario gestionar la calidad durante el desarrollo de los componentes.

1.4. Gestión de la calidad para el desarrollo de software basado en líneas de componentes

La gestión de la calidad para el desarrollo de software basado en líneas de componentes se rige por las normas definidas para el desarrollo de software, como la norma ISO 9001 y el modelo CMMI. El desarrollo basado en líneas de componentes tiene la particularidad de desarrollar diversos componentes a la vez para su posterior uso o integración en un sistema. Los componentes desarrollados han de cumplir con los requisitos funcionales definidos para los mismos, pero a su vez han de poseer determinadas características para ser considerado un componente. Por lo cual se hace necesario la realización de actividades en la etapa de desarrollo para comprobar y garantizar que los componentes cumplan con características de calidad para su posterior uso.

El estándar ISO/IEC 9126 presenta características para la evaluación de la calidad de un software o componente de software y su descomposición en sub-características. A continuación se presentan dichas características las cuales pueden brindar al usuario indicadores para medir el nivel de calidad de un componente (ISO:9126-1, 2000).

Tabla# 1 Características de la calidad según la norma ISO 9126

Características	Sub-características
Funcionalidad	Adecuación, corrección, interoperabilidad, conformidad, seguridad.
Fiabilidad	Madurez, recuperabilidad, tolerancia a fallos.
Usabilidad	Facilidad de aprendizaje, Facilidad de comprensión, operabilidad.
Eficiencia	Comportamiento temporal, utilización de recursos.
Mantenibilidad	Estabilidad, analizabilidad, cambiabilidad, Facilidad de prueba.
Portabilidad	Facilidad de instalación, adecuación, reemplazabilidad, adaptabilidad.

- **Funcionalidad:** Capacidad de proporcionar las funcionalidades que satisfacen las necesidades explícitas e implícitas cuando el sistema se usa bajo condiciones específicas.
- **Fiabilidad:** Es aplicable directamente a los componentes, y es fundamental para su reutilización. Es la capacidad de mantener un nivel especificado de prestaciones cuando se usa bajo condiciones específicas.
- **Usabilidad:** Capacidad de ser entendido, aprendido, usado y ser amigable para el usuario, cuando se usa bajo condiciones especificadas.

- **Eficiencia:** Capacidad de proporcionar prestaciones apropiadas, relativas a la cantidad de recursos usados, bajo condiciones determinadas.
- **Mantenibilidad:** Capacidad de ser modificado. Las modificaciones podrían incluir correcciones, mejoras o adaptación del software a cambios en el entorno, requisitos y especificaciones funcionales.
- **Portabilidad:** Capacidad de ejecutarse en diferentes plataformas o sistemas operativos.

El desarrollo de software basado en componentes incluye la implementación e integración de componentes previamente desarrollados. Para llevar a cabo la selección de un componente para integrarlo a un sistema, la calidad es un factor imprescindible. Cuando dos componentes presentan las mismas funcionalidades es necesario medir la calidad de cada uno de ellos para decidir cuál será utilizado. Para garantizar la calidad durante la implementación de un componente se realizan actividades con el objetivo de cumplir las características anteriormente citadas, estas actividades se denominan pruebas de software.

1.5. Pruebas de Software

Las pruebas de software son un proceso que se realiza como parte del control de la calidad, con el objetivo de aumentar la calidad del software es recomendable que sea evaluado a lo largo de su construcción. También es necesario realizar en paralelo al proceso de desarrollo, un proceso de evaluación y comprobación de los productos que se van creando. Específicamente se puede definir las pruebas de software como:

Según Vázquez, (Vázquez, 2006)*: "El proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final. Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, los resultados son observados y registrados, y una evaluación es hecha de aspectos del sistema o componente."*

Las pruebas de software tienen los siguientes objetivos:

- Encontrar y documentar los desperfectos que pueden afectar la calidad del sistema.
- Verificar que el sistema trabaje como fue diseñado.
- Validar y probar los requisitos que debe cumplir el sistema.
- Validar que los requisitos fueron implementados correctamente.

Para la aplicación de las pruebas de software es importante comprender los principios básicos que guían las mismas. Pressman propone los siguientes principios (Pressman, 2005):

- A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.
- Las pruebas deberían planificarse mucho antes de que empiecen.
- Las pruebas deberían empezar por lo pequeño y progresar hacia lo grande.
- No son posible las pruebas exhaustivas.
- Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente.

Los principios de las pruebas son las formas en que se deben aplicar las pruebas de software para hacerlas más eficaces y eficientes, aumentando de esta manera la probabilidad de encontrar errores. Existen otros conceptos importantes relacionados con las pruebas de software entre los que se encuentran:

Defecto: Un defecto es un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa (Casallas, 2005).

Fallo: La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados (Casallas, 2005).

Error: Diferencia entre el valor medido o calculado y el real. (Española, 2015)

Luego de haber analizado los aspectos relativos a pruebas de software, se puede concluir que su objetivo es encontrar el mayor número de errores con la mínima cantidad de esfuerzo y tiempo posible. Para ello existen métodos para el diseño de casos de pruebas los cuales tienen como objetivo cubrir todas las alternativas que se pueden efectuar en el sistema con la menor cantidad de pruebas.

1.6. Métodos de diseño de casos de pruebas

Los casos de prueba son situaciones que tienen un conjunto de condiciones o variables bajo las cuales el analista determinará si el sistema funciona correctamente. El diseño de casos de prueba tiene como principal objetivo obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del sistema y muestren que el software satisface sus requerimientos (Sommerville, 2007). Para alcanzar este objetivo, se usan dos métodos diferentes para el diseño de casos de prueba estas son: pruebas de caja blanca y pruebas de caja negra.

1.6.1. Prueba de Caja Blanca

La prueba de caja blanca, también conocida como caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los mismos. Con la utilización de los métodos de caja blanca se pueden obtener casos de prueba que:

- Garanticen que se ejercitan por lo menos una vez todos los caminos independientes de cada módulo.
- Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

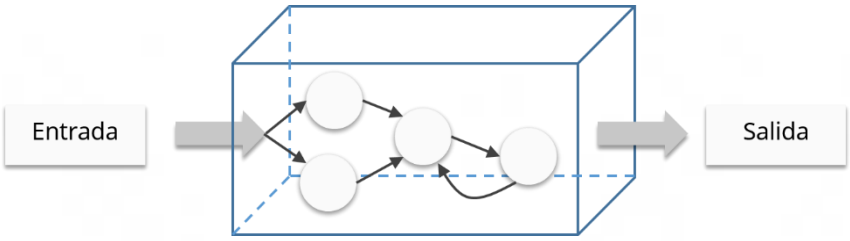


Fig. 1 *Pruebas de caja blanca. Fuente: Elaboración propia.*

Existen diferentes técnicas de prueba de caja blanca, las cuales se utilizan en dependencia del objetivo del caso de prueba que se quiera diseñar. A continuación se describen los diferentes tipos de técnicas de caja blanca existentes:

Prueba de camino básico: Esta prueba 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. El método de esta prueba consiste en derivar casos de prueba a partir de un conjunto de caminos independientes por los que puede circular el flujo de control. Los casos de prueba obtenidos garantizan que durante la prueba se ejecuten por lo menos una vez para cada sentencia del programa (Pressman, 2005).

Prueba de condición: La prueba de condición es una técnica de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. El propósito de la prueba de condiciones es detectar, no solo los errores en las condiciones de un programa, sino también otros errores en dicho programa (Pressman, 2005).

Prueba de flujo de datos: La prueba de flujo de datos selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa. Es poco realista asumir que la prueba del flujo de datos puede ser utilizada para probar grandes sistemas, esta prueba ha de ser utilizada en áreas del software que sean inseguros (Pressman, 2005).

Prueba de bucles: La prueba de bucles es una técnica de prueba de caja blanca que se centra en la validez de las construcciones de bucles.

Luego de haber analizado las variantes existentes de la prueba de estructura de control, las cuales amplían la cobertura de la prueba y mejoran la calidad de la prueba de caja blanca, se concluye que la técnica a emplear para generar los casos de prueba de la estrategia de pruebas para líneas de desarrollo de componentes propuesta es la prueba de camino básico, por su sencillez y alta efectividad (Pressman, 2005).

1.6.2. Prueba de Caja Negra

La prueba de caja negra o prueba de comportamiento, se enfoca en los requisitos funcionales del sistema, permitiéndole al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. No es una alternativa a las técnicas de prueba de caja blanca, se trata de un enfoque que intenta descubrir diferentes tipos de errores que los métodos de caja blanca no son capaces de detectar (Tenorio, 2010).

La prueba de caja negra centra la detección de errores en las siguientes categorías:

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

Fig. 2 *Pruebas de caja negra. Fuente: elaboración propia.*

La prueba de caja negra ignora intencionalmente la estructura de control y centra su atención en el campo de información. Para el desarrollo de estas pruebas existen diversas técnicas, entre las que se encuentran:

Partición equivalente: Es una técnica que divide el campo de entrada de un programa en clases de datos de los que se pueden generar casos de prueba. La partición equivalente se centra en la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba a desarrollar. Según Pressman, (Pressman, 2005): "*Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada*". Aplicando los métodos para la obtención de clases de equivalencia, se pueden desarrollar y ejecutar casos de prueba para cada elemento de datos del campo de entrada.

Análisis de Valores Límite: El análisis de valores límite (AVL) es una técnica de diseño de casos de prueba que complementa la partición equivalente. En vez de seleccionar cualquier elemento de una clase de equivalencia, el AVL traslada la selección de casos de prueba en los extremos de la clase. El AVL centra los casos de prueba tanto en las condiciones de entrada como en las de salida *(Pressman, 2005)*.

Métodos Basados en Grafos: El primer paso en la prueba de caja negra es entender los objetos de datos, módulos y colecciones de sentencias que se modelan en el sistema y las relaciones que los conectan. Luego se definen una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. La esencia de este método consiste en que un grafo representa la relación entre objeto-dato y objeto-programa, permitiendo derivar casos de prueba que buscan errores asociados a estas relaciones *(Pressman, 2005)*.

Prueba de Comparación: Las pruebas de comparación se utilizan cuando se desarrolla software en serie, en grandes cantidades y en versiones independientes al prototipo de la aplicación. Para detectar si existen errores en las versiones independientes se les aplican casos de prueba comparando si las versiones son iguales a la aplicación de software, si no lo son se deberán determinar cuáles fueron los defectos *(Pressman, 2005)*.

Prueba de la tabla ortogonal: La prueba de la tabla ortogonal puede aplicarse a problemas donde el dominio de entrada es demasiado grande para realizar pruebas exhaustivas. Se utiliza para encontrar errores asociados con fallos localizados. Esta prueba detecta y aísla los fallos de modalidad simple, detecta los fallos de modalidad doble así como los fallos multimodales *(Pressman, 2005)*.

Dentro del método de Caja Negra la técnica Partición de Equivalencia es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el sistema, descubre de forma inmediata una clase de errores que requerirían la ejecución de muchos casos antes de detectar el error genérico. Por estas características este método será el utilizado para la realización de pruebas de caja negra para la propuesta de estrategia de pruebas a desarrollar.

Para realizar la correcta construcción de un sistema informático se han de integrar métodos de diseño de casos de prueba conformados por pasos bien planificados. Algunos autores como Kruchten y Pressman afirman que el proceso de ejecución de pruebas debe ser considerado durante el ciclo de vida de un proyecto, para así obtener un producto de alta calidad. Su éxito dependerá del seguimiento de una estrategia de pruebas adecuada.

1.7. Estrategia de Pruebas

Según Pressman, (Pressman, 2005): *"La Estrategia de Pruebas de software integra un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba: la planificación, el diseño de casos de prueba, la ejecución y los resultados, tomando en consideración cuánto esfuerzo y recursos se van a requerir, con el fin de obtener como resultado una correcta construcción del software".*

La estrategia proporciona un mapa que describe los pasos que hay que llevar a cabo como parte de la prueba. Cualquier estrategia de pruebas debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de las pruebas y la agrupación y evaluación de los datos resultantes (Scalone 2006).

El International Software Testing Qualifications Board (ISTQB) plantea que una estrategia de pruebas es la descripción de los diferentes niveles de pruebas y las pruebas que deben realizarse en cada uno de ellos en una organización o conjunto de proyectos.

Partiendo de las definiciones antes expuestas se puede concluir que una estrategia de pruebas es la integración de técnicas de diseño de casos de pruebas en una serie de pasos que han de incluir la planificación, diseño de casos de prueba, ejecución y evaluación de resultados, con el objetivo de garantizar la correcta construcción del sistema.

Actualmente existen diversas estrategias de pruebas de software a nivel mundial como estrategias de pruebas para software orientado a objetos, estrategias de pruebas para líneas de productos software y estrategias de pruebas para software convencionales. Generalmente las estrategias de pruebas son

diseñadas para un producto, módulo, empresa o metodología de desarrollo de software específica, por lo que definen actividades o procesos específicos para el objetivo con que fueron diseñadas, haciéndolas ineficientes para otros fines. En la UCI también se han desarrollado múltiples estrategias de pruebas de software como: la Estrategia de Pruebas para los Software del Centro de Telemática, Estrategia de pruebas de software para el Centro de Informatización de la Seguridad Ciudadana y Estrategia de pruebas para el proyecto fuerza de trabajo calificada (Institucional, 2011). Estas estrategias también se encuentran definidas para un producto específico, por lo cual no son aplicables a líneas de componentes, pero su estudio permitió adquirir conocimientos sobre la definición y estructura de forma general.

La búsqueda permitió identificar elementos positivos para el desarrollo de la investigación, las LPS tienen un enfoque dirigido a productos, pero basado en componentes por lo que abordan varios conceptos que resultan de gran utilidad, tales como el enfoque iterativo, la generación de resultados en cortos tiempos y la reutilización interna. Por lo que se trabaja para crear y diseñar una estrategia de pruebas que permita medir y elevar la calidad durante el proceso de desarrollo de componentes de software.

1.7.1. Diseño de estrategia de pruebas.

Para el diseño de una estrategia de pruebas es fundamental la definición de aspectos comunes para toda estrategia. Inicialmente se define el propósito de la estrategia y los objetivos que se pretenden alcanzar con la misma. El alcance de la estrategia es otro aspecto imprescindible, este ha de ser definido para tener claridad del campo de acción en el que opera la estrategia.

Una vez definido el propósito, objetivos y alcance se definen las actividades de pruebas y fases que se realizan posteriormente mediante líneas de tiempo con respecto al cronograma definido en el plan de pruebas. La modelación de estas actividades y fases han de comprender los roles que intervienen en la misma, el objetivo de las actividades y los artefactos o documentos que se generan o utilizan. La descripción de las fases y actividades ha de ser clara e intuitiva por lo cual se recomienda la utilización de diagramas de procesos de negocios para la representación del flujo de trabajo de la estrategia.

El proceso de pruebas ha de describirse correctamente, definiendo los niveles de prueba a utilizar en la estrategia, roles y responsabilidades de cada miembro del equipo. También ha de definir la gestión de pruebas y las herramientas de automatización necesarias para la ejecución de las mismas. Para cada prueba definida se debe describir el objetivo, el encargado de realizar la prueba, cuando ha de realizarse la prueba, los artefactos que se generan, el enfoque de la prueba y las herramientas utilizadas para la

realización de las mismas. De forma alternativa la estrategia de pruebas también puede incluir un plan para la mitigación de riesgos así como un plan de contingencia.

Una estrategia de prueba de software debe ser flexible, para promover de esta manera la adaptabilidad necesaria para adecuar la prueba a todos los sistemas en los que vaya a ser utilizada. A su vez ha de ser lo suficientemente rígida para promover el seguimiento razonable de la planificación a medida que se realiza el proyecto (Chicago, 2007).

La planificación de una buena estrategia de pruebas es capaz de disminuir el esfuerzo para el desarrollo de las pruebas proporcionadas y reducir su tiempo de elaboración y ejecución. Las estrategias de pruebas de software sirven de apoyo al ingeniero y le brinda una plantilla para la prueba. Todas las estrategias han de cumplir con las siguientes características generales (Pressman, 2005):

- Las pruebas comienzan a nivel de módulo y trabajan hacia fuera, es decir hacia la integración del sistema.
- Se le aplica diferentes técnicas de pruebas según las que correspondan en ese momento.
- La prueba la lleva a cabo el responsable del desarrollo del sistema y para grandes proyectos un grupo independiente de pruebas.
- Las pruebas y la depuración son actividades diferentes, pero la depuración se debe incluir en cualquier estrategia de pruebas.

Cuando se evalúa dinámicamente un sistema se debe comenzar por los componentes más simples y pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto. Las pruebas se aplican durante el ciclo de desarrollo del software con diferentes objetivos y en distintos niveles de prueba entre las que se encuentran las pruebas unitarias, pruebas de integración, pruebas de validación y por último pero no por menos importante las pruebas de sistema, a continuación se describen dichos niveles de pruebas.

1.8. Niveles de Pruebas

Según Pressman el proceso de ingeniería del software se puede ver como una espiral. Inicialmente, la ingeniería de sistemas define el papel del software y conduce al análisis de los requisitos del mismo, en el cual se establece el dominio de información, la función, el comportamiento, el rendimiento, las restricciones

y los criterios de validación del sistema informático. La estrategia de software también se puede ver en el contexto de la espiral.

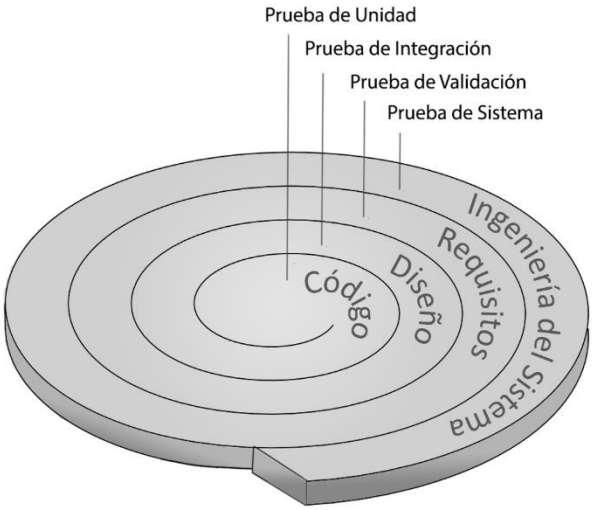


Fig. 3 *Niveles de pruebas. Fuente: Elaboración propia.*

La Fig. 3 Niveles de pruebas., muestra cómo interactúan los niveles de pruebas. El vértice de la espiral del desarrollo de software está marcado por la prueba de unidad, centrándose en cada unidad del software como nodos aislados para detectar los errores en la lógica y en la funcionalidad del programa. El siguiente paso es la realización de las pruebas de integración, en las cuales se analiza el diseño y construcción de la arquitectura del sistema, así como la unión de los nodos anteriormente citados. Llegado este punto, se realiza la prueba de validación, la cual tiene como objetivo demostrar mediante pruebas de caja negra la conformidad con los requisitos definidos por el cliente. Para concluir el proceso de pruebas se aplican las pruebas de sistema, donde se prueba como un todo el sistema desarrollado. A continuación se describen los niveles de pruebas definidos por Pressman:

1.8.1. Pruebas de Unidad

Las pruebas de unidad centran el proceso de verificación sobre el componente de software o módulo, el cual constituye la menor unidad del diseño del sistema. La interfaz del módulo se comprueba para asegurar que la información fluye correctamente. Se verifican las estructuras de datos locales con el fin de asegurar la integridad de los datos que se mantienen temporalmente durante los pasos de ejecución del algoritmo. Se prueban las condiciones límite, asegurando así que el módulo opera correctamente en los límites establecidos. Se ejercitan los caminos independientes de la estructura de control para asegurar que las

sentencias del módulo se ejecutan al menos una vez. Para concluir la prueba se examinan todos los caminos de manejo de errores.

Las pruebas deben ser realizadas por el diseñador y el programador del módulo. Luego de ser realizadas a cada módulo se procede a ensamblar o integrar los módulos para formar el paquete de software completo, proceso que se puede apreciar en la Fig. 4 Pruebas de Unidad.

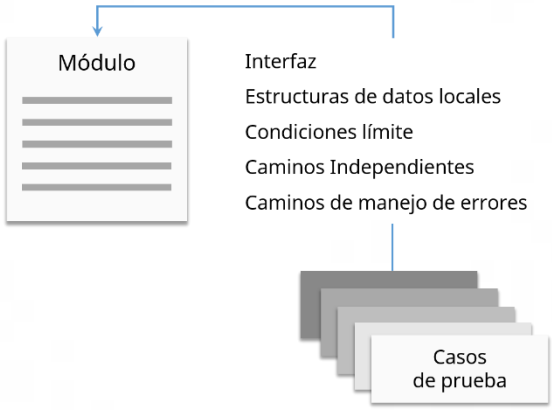


Fig. 4 Pruebas de Unidad. Fuente: Elaboración propia.

1.8.2. Prueba de Integración

Las pruebas de integración se realizan durante la elaboración del sistema, estas involucran una gran cantidad de módulos y terminan probando el sistema como un conjunto. Pueden plantearse desde un punto de vista estructural o funcional. Las pruebas estructurales son similares a las pruebas de caja blanca las cuales se explicaron en el epígrafe 1.4. Aunque en lugar de referirse a sentencias del lenguaje, hacen alusión a llamadas entre módulos. Las pruebas funcionales de integración se asemejan a las pruebas de caja negra, las cuales se explicaron en el epígrafe 1.4. En estas se intenta encontrar fallos en la respuesta de un módulo cuando su operación depende de servicios prestados por uno o varios módulos. Las pruebas finales de integración cubren todo el sistema, sin embargo, se enfocan a un alto nivel después que el software ha sido integrado.

Existen dos tipos de pruebas de integración la incremental y la no incremental. En la integración no incremental se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto. En la integración incremental se construye y se prueba en pequeños segmentos donde los errores son más fáciles de aislar y

de corregir, es más probable que se puedan probar completamente las interfaces y se puede aplicar un enfoque de prueba sistemática.

La prueba de integración que se ajusta al desarrollo de software basado en componentes es la integración incremental, ya que facilita que los componentes sean probados, maximizando la facilidad para encontrar errores en la integración de los mismos al sistema. Por este motivo para el diseño de la propuesta de la estrategia de pruebas de software para líneas de componentes se utiliza este tipo de prueba. A continuación se exponen diferentes tipos de estrategias de integración incrementales:

- **Integración descendente**: Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando por el módulo de control principal o programa principal. Los módulos subordinados o secundarios se van incorporando en la estructura, de forma primero-en-profundidad o de forma primero-en-anchura.
- **Integración ascendente:** Comienza la construcción y la prueba con los módulos atómicos. Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardos.
- **Prueba de regresión:** La prueba de regresión es volver a ejecutar un subconjunto de pruebas que se han realizado anteriormente con el objetivo de asegurar que los cambios no han propagado efectos colaterales no deseados. Ayuda a asegurar que los cambios debidos a las pruebas o por otros motivos no introducen un comportamiento no deseado o errores adicionales.
- **Prueba de humo:** La prueba de humo describe el proceso de validar cambios de código antes de que los cambios se registren en el árbol de origen del producto. Después de las revisiones de código, las pruebas de humo son el método más rentable para identificar y corregir defectos en el software. Las pruebas de humo están diseñadas para confirmar que los cambios en el código funcionan como se espera y no desestabilizan una versión completa.

1.8.3. Prueba de Validación

Luego de la realización de las pruebas de integración, el software se encuentra ensamblado como un paquete y se han encontrado y corregido los errores de interfaz, por lo cual es el momento idóneo de realizar pruebas enfocadas en los requisitos del sistema denominadas pruebas de validación. La validación del software se consigue mediante pruebas de caja negra que demuestran la conformidad con los requisitos definidos por el cliente. En conjunto las pruebas diseñadas tienen como objetivo asegurar que se satisfacen todos los requisitos

funcionales, que se alcanzan todos los requisitos de rendimiento, que la documentación es correcta y evidente y que se alcanzan otros requisitos como portabilidad, compatibilidad, recuperación de errores y facilidad de mantenimiento.

Resulta imposible que un desarrollador pueda prever como utilizará el usuario el programa, en ocasiones el usuario malinterpreta las instrucciones de uso, utiliza extrañas combinaciones de datos o nos es capaz de comprender una salida. Cuando se construye un sistema para un cliente en específico, se realizan **pruebas de aceptación** para que el cliente valide todos los requisitos. Estas pruebas las realiza el usuario final, puede tener lugar a lo largo de semanas, descubriendo así errores acumulados en el sistema. Si el software se elabora para ser utilizado por varios clientes, no es práctico realizar pruebas de aceptación para cada uno de ellos. En este caso se realizan las **pruebas alfa y beta** con el objetivo de descubrir errores que parezca que sólo el usuario puede descubrir. Las pruebas alfa se llevan a cabo por un cliente en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y los problemas de uso. Las pruebas beta se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes.

1.8.4. Pruebas de Sistema

Las pruebas de sistema están constituidas por una serie de pruebas cuyo objetivo primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas. A continuación se hace mención de los tipos de pruebas de sistema existentes:

- **Prueba de recuperación:** Es una prueba del sistema que fuerza el fallo del sistema de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente. Si la recuperación es automática hay que evaluar la corrección de la inicialización, de los mecanismos de recuperación, del estado del sistema, de la recuperación de datos y del proceso de re arranque. Si la recuperación requiere la intervención humana, hay que evaluar los tiempos medios de reparación para determinar si están dentro de los límites aceptables (Pressman, 2005).
- **Prueba de seguridad:** Intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán de accesos impropios (Pressman, 2005).
- **Prueba de rendimiento:** Está diseñada para probar el rendimiento del sistema en tiempo de ejecución dentro del contexto de un sistema integrado. La prueba de rendimiento se da durante todos los pasos

del proceso de la prueba sin embargo hasta que no están completamente integrados todos los elementos del sistema no se puede asegurar realmente el rendimiento del sistema. Con esta prueba, el encargado puede descubrir situaciones que lleven a degradaciones y posibles fallos del sistema (Pressman, 2005).

Para el desarrollo de la propuesta de estrategia de pruebas para líneas de componentes se utilizan los cuatro niveles de pruebas definidos por Pressman. Se incluye en la estrategia la aplicación de pruebas unitarias teniendo como objetivo probar las unidades de código más pequeñas del sistema, la integración ascendente con el objetivo de probar la correcta interacción entre los componentes, partiendo desde los más pequeños hasta probar el sistema en su totalidad, pruebas de validación para comprobar que el componente satisface los requisitos funcionales del sistema, mediante la realización de pruebas de aceptación o pruebas alfa y beta según sea necesario y por último se incluyen las pruebas de sistema con el objetivo de ejercitar el sistema.

1.8.5. Depuración

Las pruebas del software son un proceso que puede planificarse y especificarse sistemáticamente. Se puede llevar a cabo el diseño de casos de prueba, definición de una estrategia y evaluación de los resultados en comparación con las expectativas.

La depuración es el resultado de una prueba efectiva. Cuando un caso de prueba descubre un error, la depuración es el proceso en el cual se erradica el error. La depuración no es una prueba, pero siempre ocurre como consecuencia de una. El proceso de depuración comienza con la ejecución de un caso de prueba. Se evalúan los resultados y entonces arroja una incoherencia entre los resultados esperados y los encontrados. Intenta hacer corresponder el sistema con una causa, llevando así a la corrección del error. Siempre tiene uno de dos resultados:

- Se encuentra la causa, se corrige y se elimina.
- No se encuentra la causa.

En caso de no encontrar la causa, el encargado de realizar la depuración debe sospechar la causa, diseñar un caso de prueba que ayude a ratificar sus sospechas y nuevamente realiza la corrección del error de forma iterativa. A continuación se muestra un diagrama representativo del proceso de depuración.

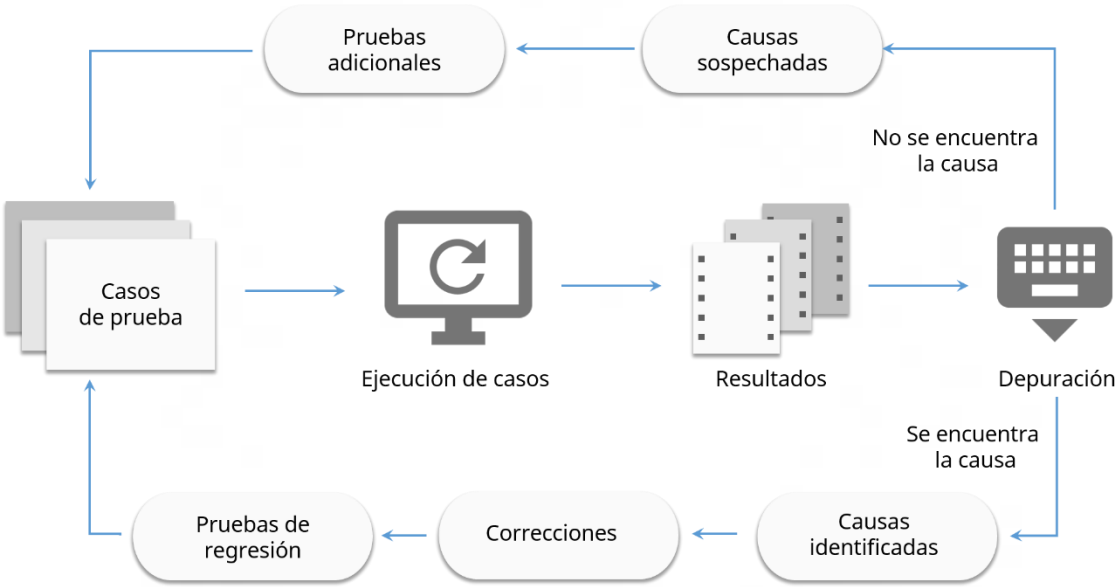


Fig. 5 *Proceso de depuración. Fuente: Elaboración propia.*

A modo de resumen se puede concluir que la depuración es el proceso de corrección de errores encontrados en el software, en este proceso se aplican los casos de prueba originando los resultados para la depuración. Luego se identifican las causas de los errores y se corrigen, dando paso a la aplicación de pruebas de corrección para conocer si se modificó el software, aplicando casos de prueba y de esta manera repetir el ciclo hasta que no se detecten errores.

1.9. Herramientas

La aplicación de las pruebas de software puede darse de forma manual o automatizada. La automatización de pruebas consiste en el uso de software especial para controlar la ejecución de pruebas y comparar los resultados obtenidos con los esperados. Permite incluir pruebas repetitivas o adicionar pruebas cuya ejecución manual resultaría difícil (Martínez, 2011).

A continuación se describen herramientas para la automatización de pruebas. Una vez analizadas estas herramientas se realizará la selección de las que serán utilizadas tanto para el desarrollo de la propuesta de la estrategia de pruebas como para el diseño de la misma. A su vez se describen las características de la herramienta de gestión de proyectos GESPRO y la herramienta para el modelado de procesos y negocios Visual Paradigm, las cuales se utilizan en las actividades correspondientes a cada una de ellas.

Web Security, es una herramienta que se utiliza para mitigar riesgos relacionados con la existencia de vulnerabilidades en aplicaciones web y la probabilidad de sufrir ataques de Hackers y usuarios malintencionados, que podrían descubrir y explotar esas vulnerabilidades causando daño al negocio. El área TI[1] provee soluciones en Pruebas de Seguridad que abordan la problemática de minimizar o mitigar la probabilidad de ocurrencia de estas vulnerabilidades que representan riesgos para la organización del cliente (www.kali.org, 2015).

Kali Linux es una distribución basada en Debian GNU/Linux diseñada principalmente para la auditoría y seguridad informática. Kali ha sido desarrollado a partir de la reescritura de BackTrack, que se podría denominar como la antecesora de Kali Linux. El cual trae preinstalados numerosos programas incluyendo Nmap (un escáner de puertos), John the Ripper (un crackeador de passwords) y la suite Aircrack-ng. Kali puede ser usada desde un Live CD, live-usb y también puede ser instalada como sistema operativo principal (www.kali.org, 2015).

Nmap ("Network Mapper") es una herramienta gratuita para la exploración de la red diseñada con el objetivo de analizar rápidamente grandes redes, aunque funciona correctamente contra equipos individuales. Nmap utiliza paquetes IP para determinar qué hosts están disponibles en la red, qué servicios (nombre de la aplicación y la versión) ofrecen estos equipos, qué sistemas operativos (y versiones del sistema operativo) se están ejecutando y qué tipo de filtros de paquetes o cortafuegos están en uso. Nmap se ejecuta en la mayoría de los ordenadores y la consola y versiones gráficas están disponibles. Nmap es libre y de código abierto (www.kali.org, 2015).

La extensión de Firefox **Web Development** es una herramienta para los desarrolladores web. Es una extensión para los navegadores web basados en Mozilla Firefox la cual provee utilidades de edición y depuración para desarrolladores web. Ha sido probada para ser compatible con los navegadores Mozilla Firefox, Flock y Seamonkey (Hammond, 2015).

[1] TI: Tecnología e innovación.

AdventNet QEngine Web Test es una aplicación que posee una amplia gama de herramientas independientes de la plataforma, para pruebas de funcionalidad y de carga de la Red en aplicaciones web desarrolladas usando HTML, JSP, ASP, .NET, PHP y JavaScript/VBScript. La herramienta de pruebas de Red ha sido desarrollada en Java, lo que facilita su portabilidad, soporte para múltiples plataformas (Windows, Linux, y Solaris) y para múltiples navegadores (Espinoza, 2005).

Selenium IDE, es una extensión de Firefox que permite escribir test de Selenium con las interacciones del usuario y ejecutarlos directamente desde el navegador. A esta herramienta se le pueden indicar rutinas de navegación para luego ejecutarlas una y otra vez y detectar así, de una manera sencilla, posibles errores. Selenium es compatible con una gran variedad de lenguajes de programación, entre los que se encuentran: Java, Python, Ruby, Perl, C # y PHP. Una de las opciones más interesantes del IDE de Selenium es la exportación de los casos de test y/o test suites a diferentes formatos (adimedia.net, 2014).

VTest es una herramienta automática para pruebas funcionales y de regresión en aplicaciones web. Incorpora funcionalidades de registro, verificación y reporte. Para su utilización no es necesario tener conocimientos de programación. Para aquellos usuarios que desean escribir programas, éste usa JavaScript como lenguaje de programación. Soporta tanto Microsoft Internet Explorer como Mozilla Firefox y se encuentra bajo licencias comerciales (Huerta, 2006).

QALoad es una herramienta de pruebas para aplicaciones que estén sometidas a gran carga de usuarios. Los sistemas cliente/servidor de hoy en día necesitan mejorar para permitir el uso concurrente a miles de usuarios. Las organizaciones necesitan desarrollar pruebas de carga repetibles y determinar el rendimiento real y límites potenciales del sistema. QALoad ayuda a alcanzar cargas que simulan el comportamiento real del negocio así como a validar que el sistema cumple aceptablemente con los niveles de servicio. Esta es una herramienta que constituye un software privativo, por tanto se precisa pagar para su adquisición, y por consiguiente por su documentación (Deris, 2005).

La herramienta proporciona:

- Pruebas escalables (Puede emular la carga generada por cientos o miles de usuarios en la aplicación)
- Fácil desarrollo de los scripts de prueba (Ofrece módulos configurables que facilitan el desarrollo de los scripts de prueba)
- Análisis exhaustivo (ofrece las estadísticas de rendimiento en tiempo real, y permite insertar puntos

de control dentro de los scripts para identificar y revisar áreas específicas del rendimiento del sistema).

LoadRunner es una herramienta para realizar pruebas de carga que permite prever el comportamiento y el rendimiento del sistema. Posibilita poner a prueba toda la infraestructura corporativa para identificar y aislar los posibles problemas mediante la simulación de la actividad de miles de usuarios. Es la herramienta de pruebas de carga más escalable que permite simular la actividad de miles de usuarios con los mínimos recursos de hardware. Facilita que los scripts creados durante las pruebas puedan volverse a utilizar para monitorizar la aplicación una vez terminada su implantación. A pesar de tener todas esas características favorables, su adquisición es sólo gratuita por 15 días, después de los cuales se debe pagar por su uso. Unido a esto la documentación está sujeta a las mismas restricciones (Deris, 2005).

JMeter es una herramienta libre, además es una aplicación desarrollada en Java, que permite realizar pruebas de rendimiento y funcionales sobre aplicaciones web. Es una herramienta de carga que permite realizar simulaciones sobre cualquier recurso de software. De las herramientas gratis, es la más completa y útil para realizar pruebas de carga. Muestra los resultados de las pruebas en una amplia variedad de informes y gráficas. Además facilita una rápida detección de los cuellos de botella existentes debido al tiempo de respuesta excesivo (Tenorio, 2010).

Se utiliza para realizar pruebas de regresión en aplicaciones web. Tiene una estructura en árbol que le da potencia, permitiendo que sea la imaginación de quien la use la que ponga los límites a la hora de diseñar el plan de pruebas. A su vez brinda mayor cantidad de variantes para recoger los resultados obtenidos, que el resto de las herramientas gratis, lo que permite hacer un análisis exhaustivo de las pruebas realizadas.

GESPRO. Es un sistema para el desarrollo y la innovación en gestión de proyectos. Es desarrollado por el Laboratorio de Investigaciones en Gestión de Proyectos de la UCI. Esta Suite de gestión de proyectos se presenta como un modelo de negocios basado en servicios donde se combina el uso de una solución informática para la gestión de proyectos y un sistema de formación especializada en gestión de proyectos. Esta combinación posibilita no sólo la informatización de la gestión de proyectos en las organizaciones, sino también la mejora continua de sus procesos de planificación, seguimiento y control (Rojas, 2004).

Visual Paradigm en su versión 8.0 para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Los sistemas de modelado UML ayudan a la construcción de aplicaciones de forma rápida, con

calidad y a bajos costos. Permite dibujar todos los tipos de diagramas de clases, ingeniería inversa, generar código a partir de diagramas, generación de objetos a partir de bases de datos, generación de bases de datos a partir de diagramas de entidad relación, generar documentación y además posee licencia gratuita y comercial.

Para realizar la selección de las herramientas a utilizar para el desarrollo y diseño de la estrategia de pruebas se tuvieron en cuenta diferentes criterios, entre los que se encuentran:

- Funcionalidades de la herramienta: Prestaciones, elementos funcionales, respuesta a los requerimientos del sistema.
- Modo de adquisición: Si el software se obtiene por medio de una licencia comercial o es gratuito.
- Materiales de apoyo: Si existe documentación de la herramienta que ofrezca una guía para su utilización.
- Entorno en el que fue desarrollado: Si es un software de código abierto o un software propietario.

Basándose en estos criterios se seleccionó para la realización de pruebas de seguridad el sistema operativo Kali Linux y Web Security. JMeter fue la herramienta seleccionada para la realización de pruebas de rendimiento. Para validar si el comportamiento del software probado cumple o no con las especificaciones del cliente se seleccionó la herramienta Selenium IDE. Para la gestión de proyectos se utilizará el sistema GESPRO y para el diseño de la propuesta de estrategia se utilizara la herramienta Visual Paradigm.

1.10. Conclusiones parciales

El estudio realizado en el presente capítulo permitió abordar temas como el análisis de la calidad de software y gestión de calidad de software en la UCI, el desarrollo de software basado en líneas de componentes, niveles, métodos y tipos de pruebas. A partir del estudio realizado se definió para la aplicación de las pruebas en la estrategia propuesta, la utilización de métodos para el diseño de casos de prueba, utilizando dentro de los métodos de Caja Negra las técnicas de camino básico y valores límites y en los métodos de Caja Blanca la técnica partición equivalente. Se especificó para el desarrollo de la estrategia de pruebas para líneas de componentes la utilización de los niveles de pruebas unitarias, de integración, de validación y de sistema definidos por Pressman. Además se seleccionaron las herramientas Kali Linux, Web Security y JMeter para la automatización de las pruebas, la herramienta GESPRO para la gestión de las no conformidades y la herramienta CASE Visual Paradigm para modelar los diagramas del diseño de la estrategia.

Capítulo 2: Estrategia de pruebas para líneas de componentes

El objetivo de este capítulo es describir la estrategia de pruebas de software propuesta para el proceso de desarrollo de líneas de componentes de software en la UCI. La estrategia de pruebas será adaptable a todas las metodologías de desarrollo de software, pero como en la universidad se trabaja con la metodología AUP-UCI a continuación se identifican actividades durante el proceso de pruebas, los artefactos utilizados para llevar a cabo las mismas, sus objetivos y sus responsables dentro de la metodología usada.

2.1. Descripción de la estrategia

Para la realización de la estrategia de pruebas es fundamental la definición de una serie de pasos bien planificados para la correcta construcción del software. La estrategia de pruebas para líneas de componentes está conformada por seis fases.

La planificación de pruebas es la primera fase que se ejecuta, en esta se define el Plan de Pruebas. Luego prosigue la fase de diseño de pruebas, en la que se planifican los casos de prueba, que serán utilizados en la siguiente fase. La ejecución es la fase en la cual se ejecutan los casos de pruebas diseñados, realizando pruebas unitarias, de integración, validación y de sistema. La ejecución de estas pruebas genera un registro de pruebas por cada prueba aplicada, los cuales serán evaluados en la fase de evaluación de resultados de pruebas, generando así un Informe de Pruebas. De existir no conformidades se procede a la fase de depuración, en la cual se localizan y corrigen los errores detectados para luego dar paso a la fase de ejecución. En caso de no detectar no conformidades en la fase de evaluación de resultados, se procede a la fase de cierre, en la cual se genera el informe de cierre de prueba, dando por culminada la estrategia. A continuación se muestra la estructura básica para la estrategia de pruebas para líneas de componentes de software.

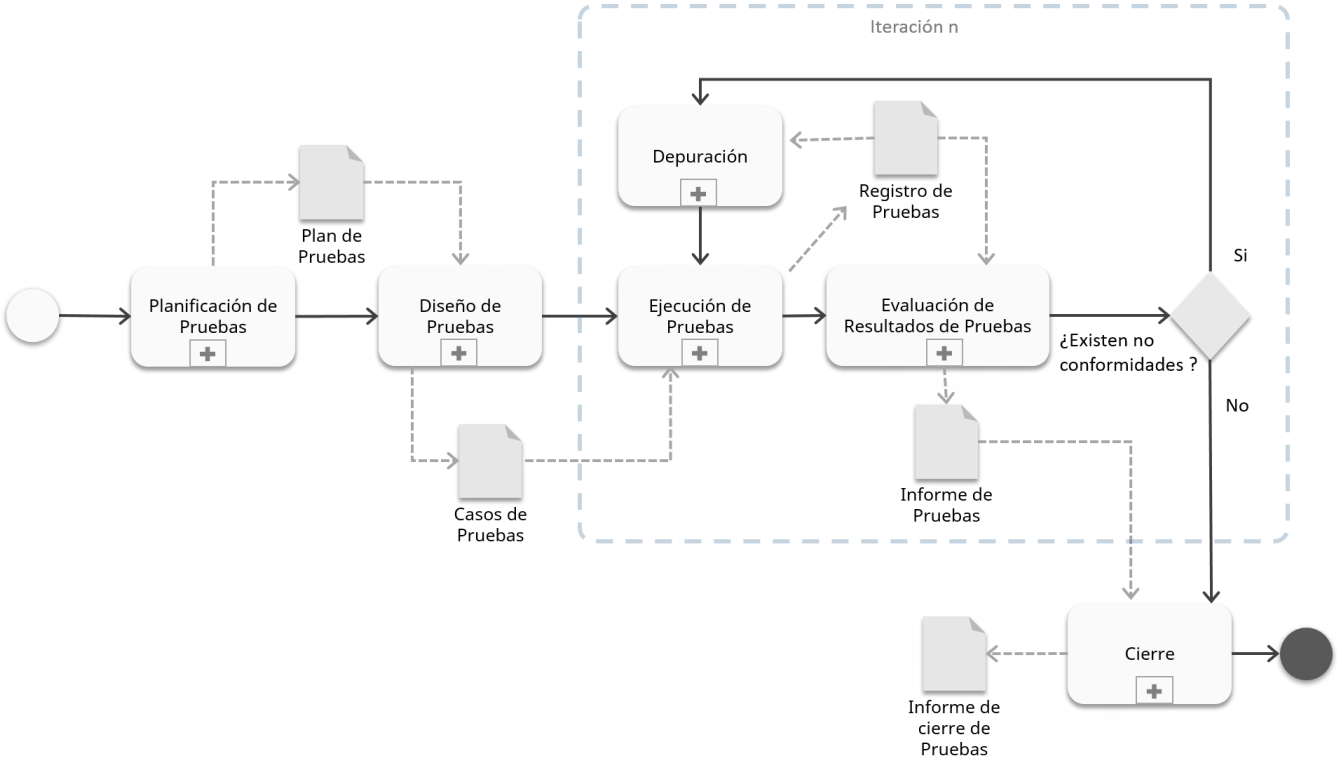


***Fig. 6** "Estrategia de Prueba". Fuente: Elaboración propia.*

2.1.1. Fase de Planificación.

El flujo del proceso de pruebas inicia con la fase de planificación de pruebas. Esta fase tiene como objetivo planificar las pruebas y actividades de calidad que se desarrollarán en la estrategia de pruebas. La primera actividad llevada a cabo es la realización del Plan de Pruebas, en la que el administrador de la calidad define el plan de pruebas, documento en el cual se definen los aspectos fundamentales que guiarán la estrategia, el alcance de las pruebas así como sus objetivos. Seguidamente el analista llevará a cabo la verificación del plan de pruebas mediante la utilización de la lista de chequeo "Verificación del Plan de Pruebas" con el objetivo de garantizar que este artefacto se halla definido correctamente. En caso de encontrar no conformidades se procede a redefinir el Plan de Pruebas, para luego verificarlo nuevamente, estas operaciones se realizan iterativamente hasta que el plan de pruebas quede libre de no conformidades.

Una vez libre de no conformidades el plan de pruebas se realiza la comprobación de las especificaciones de requisitos y la especificación de casos de uso. En esta actividad el arquitecto de software, el jefe del proyecto y el administrador de la calidad garantizan que los requerimientos se encuentren correctamente

agrupados y definidos de acuerdo al propósito o lógica del componente a desarrollar. En caso de que se encuentren no conformidades en las comprobaciones realizadas, el analista procede a rectificar los errores encontrados en los documentos revisados. Una vez corregidos los errores se procede nuevamente a aplicar la lista de chequeo, realizando este proceso de forma cíclica hasta que los documentos queden libres de errores. Seguidamente el administrador de la calidad realiza la verificación de los atributos del componente para garantizar que sea identificable, reemplazable, independiente de la plataforma, accedido solamente a través de sus interfaces, que se encuentre auto contenido y que sus servicios no varíen. Para esta actividad se comprueba el diagrama de clases y el diagrama de componentes mediante la lista de chequeo "Verificación de atributos del componente". En caso de encontrar no conformidades el analista procede a rectificar las mismas en el artefacto donde se haya detectado la no conformidad, para luego volver a aplicar la lista de chequeo, hasta que se cumpla con las pautas definidas en la misma. En la Fig. 7 Fase de planificación, se muestra el flujo de trabajo que se desarrolla en la fase de planificación.

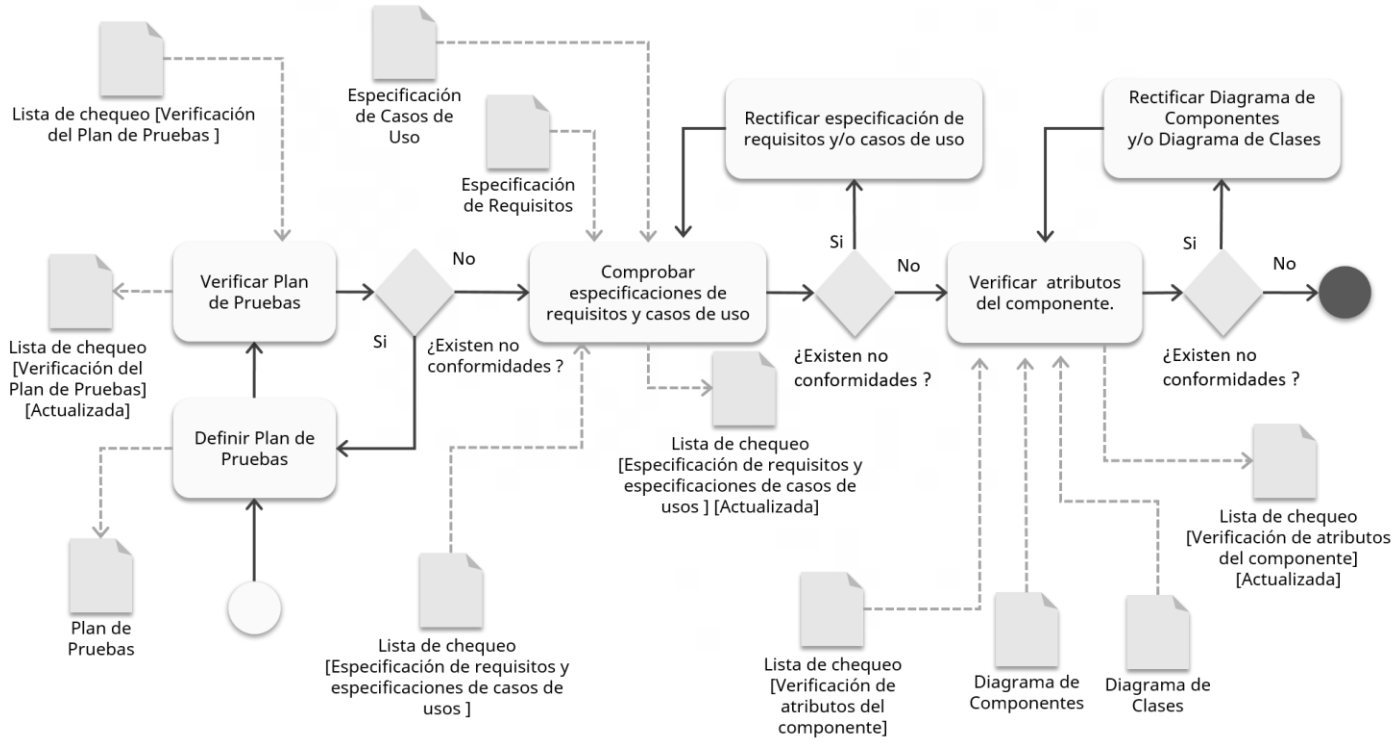


Fig. 7 *Fase de planificación. Fuente: Elaboración propia.*

2.1.2. Fase de Diseño

A continuación tiene lugar la fase de diseño de pruebas, en esta se identifica, describen y diseñan los casos de prueba para validar el componente. Luego de identificar las pruebas, se procede a la selección de las técnicas que se deben utilizar para el tipo de componente con el que se trabaja. La primera actividad de la fase es el diseño de los casos de pruebas unitarias, los cuales son elaborados por el analista y el desarrollador de software, mientras que las pruebas de integración y las pruebas de validación serán diseñadas por el analista del software. En la Fig. 8 Fase de diseño, se muestra el flujo de trabajo desarrollado en la fase de Diseño.

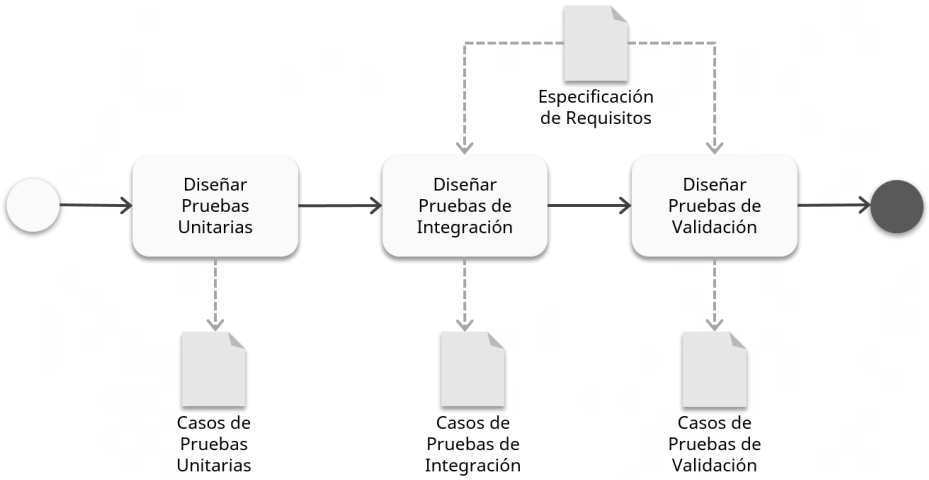


Fig. 8 *Fase de diseño. Fuente: Elaboración propia.*

2.1.3. Fase de Ejecución.

Seguidamente se efectúa la fase de Ejecución de pruebas donde se ejecutan los casos de pruebas diseñados en la fase anterior, para de esta manera validar el funcionamiento del componente. Es importante destacar que seguidamente a la realización de cada prueba, el encargado de realizar la prueba en cuestión, debe registrar las no conformidades detectadas en la herramienta GESPRO para garantizar de esta forma el tratamiento de las mismas. La primera actividad que se realiza en la fase es la aplicación de las pruebas unitarias por parte del desarrollador y el probador, realizando con estas el registro de pruebas unitarias. Una vez realizadas las pruebas unitarias, el administrador de la calidad procede a la verificación de atributos del componente a través de la lista de chequeo “Verificación de atributos del componente”, con el fin de garantizar que no hayan variado los servicios e interfaces inicialmente definidas. En caso de existir no

conformidades el desarrollador es el encargado de corregir las no conformidades en el componente, con el fin de que este se ajuste a lo definido en el diagrama de componente.

Luego de probar el componente del sistema de manera independiente, se procede a la realización de las pruebas de integración, en las que el arquitecto y el desarrollador ejercitan los casos de prueba de integración generando así el registro de pruebas de integración. Una vez finalizadas las pruebas de integración el administrador de la calidad comprueba a través de la lista de chequeo “Verificación de atributos del componente” que el componente pueda ser cargado en tiempo de ejecución. En caso de encontrarse no conformidades en la aplicación de la lista de chequeo se actualiza la lista de chequeo, para posteriormente solucionar dichos inconvenientes.

Seguidamente se llevan a cabo actividades relativas a las pruebas de validación. Para dar inicio a la fase, el probador apoyándose en los casos de prueba de validación y las especificaciones de requisitos realiza las pruebas funcionales, generando con estas el registro de Pruebas Funcionales. A continuación se procede a realizar las pruebas de aceptación o las pruebas Alfa y Beta en dependencia de si el producto en desarrollo es para uno o para varios clientes. En caso de que el producto sea para un cliente se realizan las pruebas de aceptación. En esta actividad el jefe de proyecto en conjunto con el cliente verifica que se le hayan dado cumplimiento a los requisitos funcionales del componente, generando de esta manera el Registro de pruebas de aceptación y en caso de que el cliente quede satisfecho, este firma el acta de aceptación. En caso de que el producto sea para más de un cliente, se procede a realizar las pruebas Alfa y Beta, por parte del desarrollador y uno de los clientes. Para concluir el subproceso de las pruebas de validación el administrador de la calidad realiza el registro de las pruebas de validación, el cual es el artefacto en el que se encuentran todos los artefactos generados en la aplicación de las pruebas de validación.

A continuación el probador realiza diferentes pruebas de sistema, con el fin de verificar el rendimiento, la seguridad y la recuperación del sistema, generando el registro de pruebas de sistema. Para finalizar esta fase el administrador de la calidad realiza el registro de pruebas, artefacto que contiene los registros de todas las pruebas realizadas en esta fase.

Cada una de estas pruebas arroja un artefacto con los resultados de la misma, los cuales serán evaluados en la fase evaluación de los resultados de pruebas. En la Fig. 9 Fase de ejecución, se muestra el flujo de trabajo que se desarrollará en la fase y en la figura 10 se especifica el subproceso “Realizar Pruebas de Validación”.

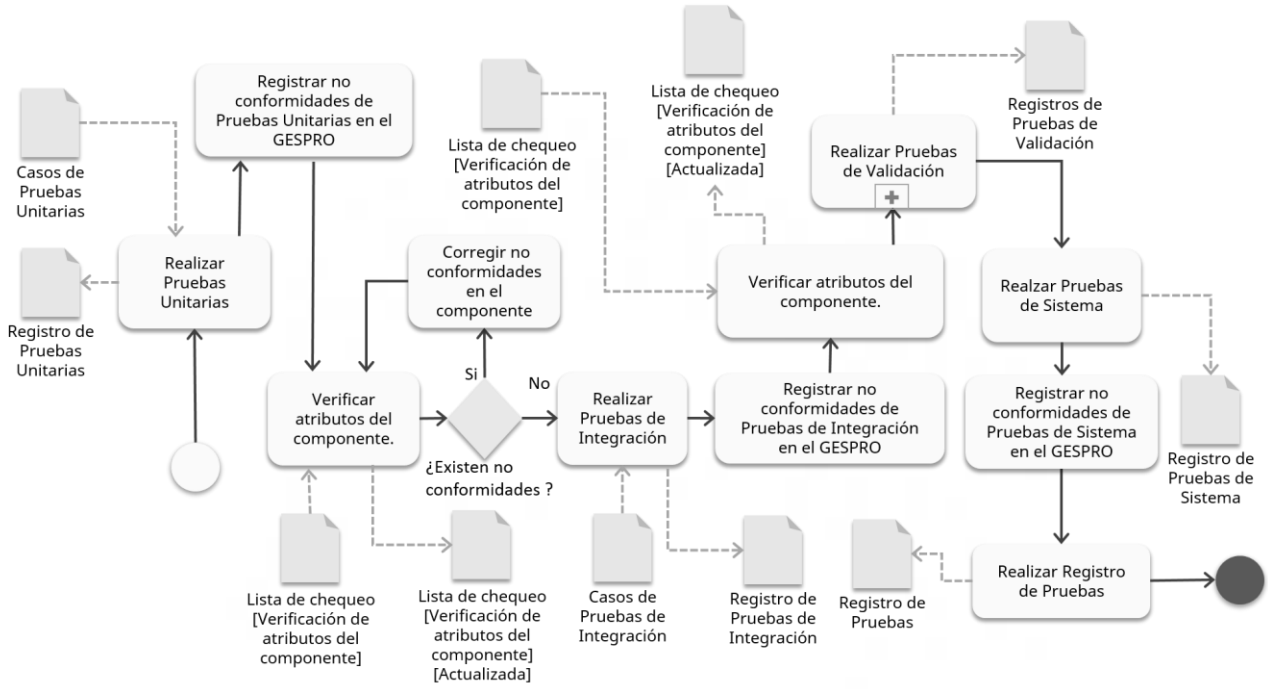


Fig. 9 *Fase de ejecución. Fuente: Elaboración propia.*

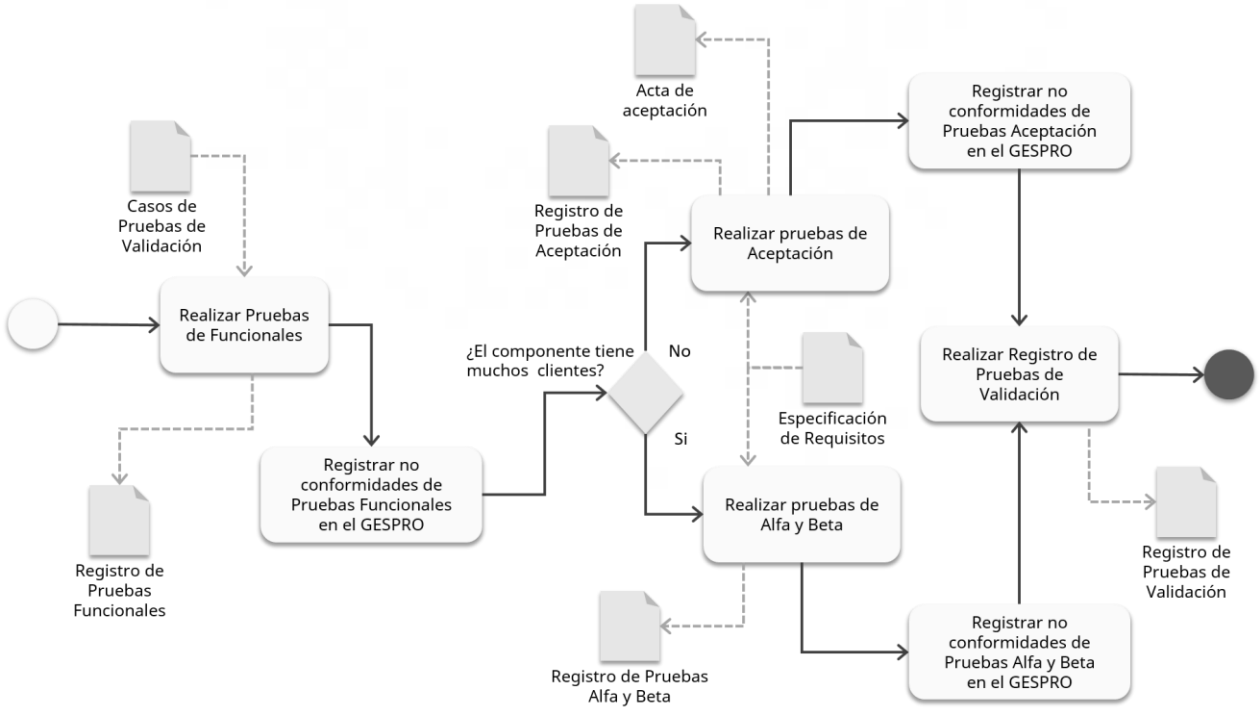


Fig. 10 *Subproceso Realizar Pruebas de Validación. Fuente: Elaboración propia.*

2.1.4. Fase de Evaluación de resultados.

En esta fase el probador analiza el registro de pruebas, para generar el informe de pruebas. Este informe contiene un resumen de los resultados obtenidos en cada nivel de prueba definido, haciendo énfasis en las existencia de no conformidades en cada uno de estos niveles. A continuación de esta evaluación y con el objetivo de verificar el cumplimiento de los indicadores de aceptación definidos para cada prueba aplicada, el analista comprueba los indicadores definidos para evaluar la calidad del producto o proceso en la línea, reflejando esta comprobación en el artefacto “Evaluación de los resultados de las pruebas”. En caso de existir no conformidades, se efectúa la fase de depuración en la que se solucionan las mismas y en caso contrario se procede a la fase de cierre. En la Fig. 11 Fase de evaluación de resultados, se muestra el flujo de la fase.

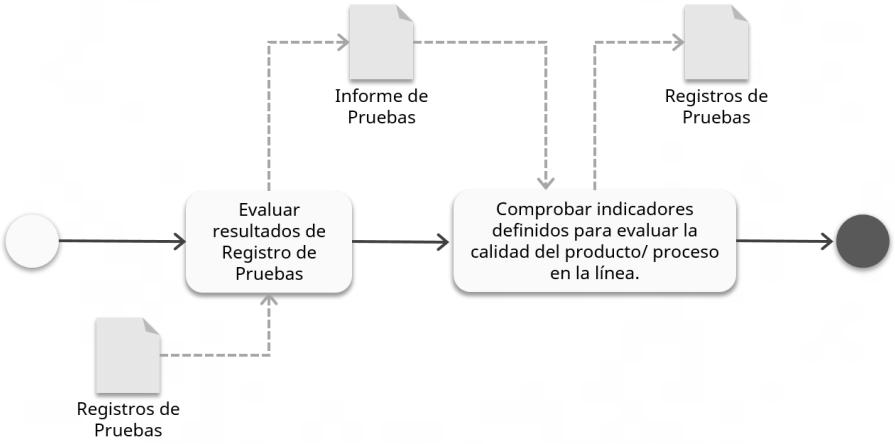


Fig. 11 *Fase de evaluación de resultados. Fuente: Elaboración propia.*

2.1.5. *Fase de Depuración.*

Es la actividad que se realiza cuando un caso de prueba descubre un error, la cual tiene como objetivo principal la erradicación de dicho error. Primeramente el desarrollador procede a depurar el sistema apoyándose en los registros que contienen las no conformidades encontradas en la fase de ejecución de las pruebas, en caso de no detectar la causa de los errores el encargado de la depuración ha de suponer donde puede encontrarse el error y en base a esta deducción procede a actualizar los casos de pruebas para que sean analizados nuevamente. En caso de detectar las causas de los errores, se procede a corregir los mismos y para asegurar que los cambios realizados no hayan afectado otra área del sistema lleva a cabo una prueba de regresión. Si en la prueba de regresión se detecta que los cambios realizados

provocaron errores en otras áreas del componente, se procede a corregir estos errores y se realiza nuevamente la prueba de regresión, realizando este proceso de forma cíclica hasta que no se detecten errores.

Luego de ser depurado el componente se procede a realizar la fase de ejecución y la fase de evaluación de resultados, conformando de esta manera un ciclo, el cual itera mientras el sistema no cumpla con los criterios de aceptación definidos para cada prueba. En la Fig. 12 Fase de depuración, se muestra el flujo de trabajo que se desarrolla en la fase de Depuración.

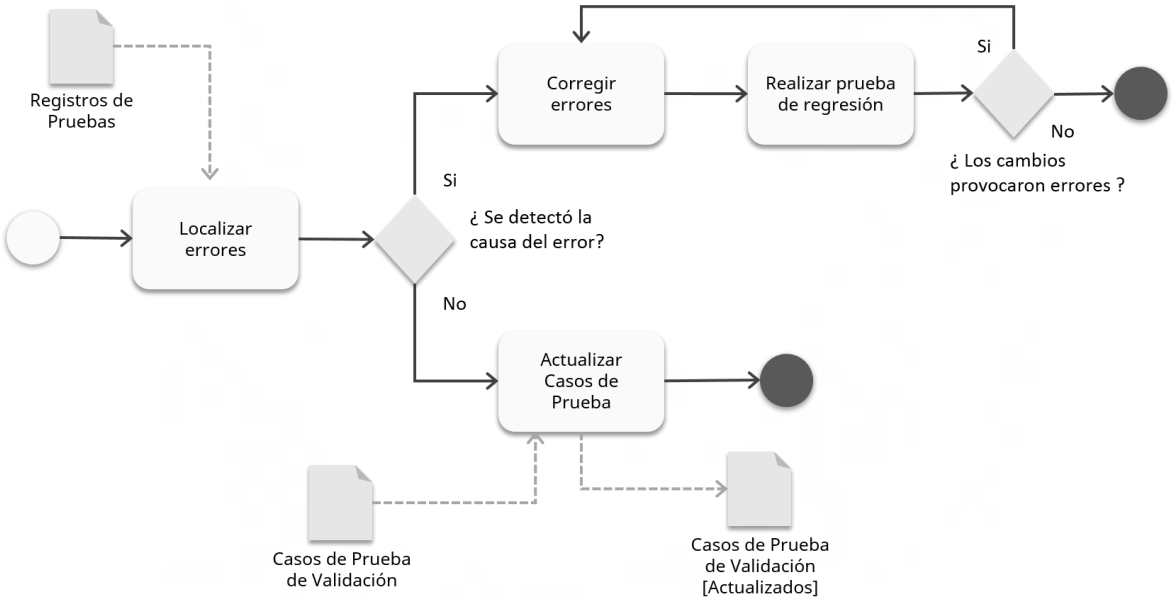


Fig. 12 *Fase de depuración. Fuente: Elaboración propia.*

2.1.6. Fase de Cierre.

En caso de que el proceso de evaluación de resultados sea satisfactorio se procede a la fase de cierre. En esta fase se concluye el proceso de pruebas. La primera actividad de la fase la realiza el administrador de la calidad, en la cual chequea la documentación del componente mediante la aplicación de la lista de chequeo para la revisión de la documentación del componente. En caso de encontrar no conformidades el analista procede a corregir la documentación del componente, para luego volver a aplicar la lista de chequeo y verificar que ya no existan no conformidades. En caso de que no se encuentren no conformidades, el analista genera el Informe de cierre de pruebas, en el cual se refleja el análisis de los errores encontrados durante la aplicación de la estrategia, así como la definición de acciones preventivas para la futura mejora

del proceso. En la Fig. 13 Fase de cierre, se muestra el flujo de trabajo que se desarrolla en la fase de Cierre.

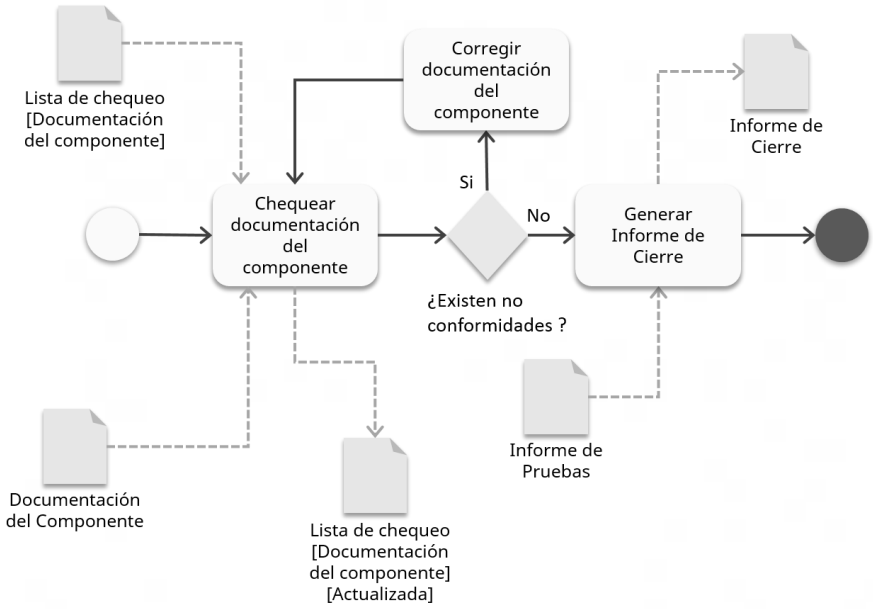


Fig. 13 *Fase de cierre. Fuente: Elaboración propia.*

2.1.7. Metas que se persiguen

Establecer buenas prácticas para el desarrollo basado en componentes. Aumentar la calidad y el nivel de reutilización de los componentes desarrollados. Disminuir los problemas en la integración de productos, ya que los componentes que lo forman estarán previamente validados. Garantizar la medición y mejora de los indicadores de calidad en la línea de componentes.

2.2. Técnicas de pruebas utilizadas

En el desarrollo de la estrategia de pruebas se utilizaron diferentes técnicas de pruebas, entre ellas se encuentran las listas de chequeo para la validación y revisión de la documentación y las pruebas unitarias, de integración, de validación y de sistema para la correcta validación del componente. A continuación se explican las técnicas utilizadas.

2.2.1. Aplicación de listas de chequeo

Las listas de chequeo son artefactos que tienen un conjunto de parámetros a medir sobre un aspecto determinado, dígase documentación o aplicación. Es un instrumento de medición y evaluación que consiste

básicamente en un formulario de preguntas referentes a los atributos de calidad que se están probando y de las características del documento en el caso de la documentación. Cada pregunta tiene asociada una evaluación en una escala que da una medida del grado de cumplimiento y disponibilidad de la propiedad evaluada, de esta manera se determina la evaluación del elemento probado (Figueredo, 2013).

Para la verificación de la correcta redacción de la documentación se utilizarán listas de chequeo por cada documento revisado tales como, Especificación de requisitos y Descripción de los casos de uso. Se tuvo en cuenta aspectos significativos como:

Estructura del Documento: Abarca todos los aspectos definidos por el expediente de proyecto o el formato establecido por el proyecto.
Elementos definidos por la metodología: Abarca todos los indicadores a evaluar según la metodología.
Semántica del documento: Contempla todos los indicadores a evaluar respecto a la ortografía y redacción.
Correspondencia del producto con los atributos de un componente de software: Se identificó en cada fase los atributos identificativos de un componente.
Objetivo de las listas de chequeo: Validar que la documentación generada en cada fase se encuentre correctamente redactada y estructurada.
Criterio de aceptación: La documentación está completa y se encuentra bien redactada y estructurada.
Responsables de ejecutar las listas de chequeo: Administrador de la calidad.
Fecha de inicio de la prueba: (Fecha de inicio de la prueba).
Fecha de fin de la prueba: (Fecha de fin de la prueba).

2.2.2. Aplicación de las pruebas unitarias

Para evaluar el funcionamiento del sistema el desarrollador realiza las actualizaciones a los diseños de casos de pruebas y registra las no conformidades encontradas, haciendo uso de la herramienta de trabajo GESPRO, donde se plasman evidencias de las deficiencias encontradas. La estrategia contempla la aplicación de pruebas de caja blanca, y dentro de las mismas se aplicará la técnica camino básico, por parte del probador y el desarrollador de software.

Dando conclusión a este nivel de prueba, se confecciona el “Registro de pruebas unitarias”, donde se recogerán los resultados obtenidos en la aplicación de las pruebas y se procede a la aplicación de las pruebas de integración.

Objetivos de la prueba: Lograr que cada módulo o servicio del componente que se desarrolla se encuentre libre de errores.
Técnica que se aplica: Se ejecutará específicamente la técnica de caja blanca camino básico.
Criterio de aceptación: Todos los casos de prueba arrojaron resultados satisfactorios y del 60%- 80% del código fue probado.
Responsables de ejecutar la prueba: Desarrollador y probador.
Fecha de inicio de la prueba: (Fecha de inicio de la prueba).
Fecha de fin de la prueba: (Fecha de fin de la prueba).

2.2.3. Aplicación de las pruebas de integración

Las pruebas de integración se realizan con el objetivo de validar que los módulos que conforman el sistema interactúen correctamente entre sí. Para ello se utiliza la técnica de prueba integración ascendente, comenzando por el módulo o servicio más pequeño.
Para la aplicación de las pruebas de integración el analista confecciona los casos de prueba, y el desarrollador en conjunto con el arquitecto de software las aplica a medida que culmine el desarrollo de cada módulo. Las pruebas se comienzan aplicando a dos módulos o servicios básicos y concluye cuando ha sido probado el componente en su totalidad.
Al concluir la ejecución de las pruebas se registran las no conformidades que se hayan detectado en el registro de pruebas de integración. Además se itera tantas veces como sea necesario hasta lograr que el componente quede libre de errores y funcione correctamente como un todo. Al concluir la prueba se confecciona el ¨Registro de pruebas de integración¨, en el cual se almacenan los datos relacionados con la aplicación de las pruebas y los resultados que arrojaron las mismas.

Objetivos de la prueba: Lograr que el componente funcione correctamente como un todo.
Técnica que se aplica: Integración ascendente.
Criterio de aceptación: Todos los casos de prueba arrojaron resultados satisfactorios (100%).
Responsables de ejecutar la prueba: Desarrollador y Arquitecto de software.
Fecha de inicio de la prueba: (Fecha de inicio de la prueba).
Fecha de fin de la prueba: (Fecha de fin de la prueba).

2.2.4. Aplicación de las pruebas de validación

Luego de realizar las pruebas de integración, el software se encuentra ensamblado como un paquete y ya se detectan y corrigen los errores de interfaz, por lo que se pueden realizar una serie de pruebas finales al software, denominadas pruebas de validación. Para la automatización de esta prueba se utiliza la herramienta informática Selenium IDE. La validación del software se consigue mediante pruebas de caja negra que demuestran la conformidad con los requisitos del sistema o también las podemos llamar **pruebas funcionales**, para almacenar los resultados de estas pruebas se genera el registro de pruebas funcionales.

Objetivos de la prueba: Validar los requisitos funcionales del componente.
Técnica que se aplica: Caja negra (partición equivalente).
Criterio de aceptación: El probador no encontró errores.
Responsables de ejecutar la prueba: Probador.
Herramienta para automatización de la prueba: Selenium IDE
Fecha de inicio de la prueba: (Fecha de inicio de la prueba).
Fecha de fin de la prueba: (Fecha de fin de la prueba).

Las pruebas de validación a desarrollar pueden ser pruebas de aceptación o pruebas alfa y beta en dependencia de la cantidad de clientes que posea el sistema desarrollado.

2.2.4.1. Pruebas de aceptación

Si el software desarrollado es un producto que va a ser usado por un solo cliente, se realizan pruebas de aceptación formales. Para este tipo de pruebas se genera el artefacto “Registro de pruebas de aceptación”. Las mismas son realizadas por el cliente en conjunto con el jefe del proyecto, en caso de encontrar errores estos son registrados haciendo uso de la herramienta GESPRO. En caso contrario se da la prueba por concluida.
Objetivos de la prueba: Se persigue que el cliente sea capaz de verificar que obtuvo el componente que solicitó.
Criterio de aceptación: El cliente no encontró errores, en caso contrario se aplicará varias veces la prueba hasta lograr que el cliente quede satisfecho.
Responsables de ejecutar la prueba: Cliente.
Fecha de inicio de la prueba: (Fecha de inicio de la prueba).
Fecha de fin de la prueba: (Fecha de fin de la prueba).

2.2.4.2. Pruebas Alfa y Beta

Si no existe un único cliente se aplican las pruebas alfa y beta, las mismas se aplican con el objetivo de verificar que el sistema cumple con los requerimientos del cliente. Para aplicar este tipo de pruebas se genera el artefacto “Plan de pruebas alfa y beta”, donde se almacenan todos los datos sobre la prueba y el producto que se está probando.

En la aplicación de las pruebas alfa, el cliente en el lugar de desarrollo verificará que el componente cumple con los requerimientos especificados al inicio de la fase de ejecución en presencia del desarrollador, quien en caso de existir errores registrará los mismos con la ayuda de la herramienta GESPRO y les dará solución, con el objetivo de que la prueba sea ejecutada nuevamente, repitiendo el proceso hasta lograr que el producto quede libre de errores y el cliente quede satisfecho.

Objetivos de la prueba: Se persigue que el cliente sea capaz de verificar que obtuvo el componente que solicitó.

Criterio de aceptación: El cliente no encuentra errores, en caso contrario se aplicará tantas veces como sea necesario la prueba hasta lograr que el cliente quede satisfecho.

Responsables de ejecutar la prueba: Cliente, en presencia del desarrollador.

Fecha de inicio de la prueba: (Fecha de inicio de la prueba).

Fecha de fin de la prueba: (Fecha de fin de la prueba).

2.2.5. Aplicación de las pruebas de sistema

Se aplican las pruebas de seguridad, con el objetivo de penetrar la seguridad implementada en el sistema, evidenciando la existencia de fallas de seguridad y por otra parte se aplican las pruebas de rendimiento con el objetivo de medir el rendimiento del sistema en situaciones inauditas.

2.2.5.1. Aplicación de las pruebas de rendimiento.

En la aplicación de las pruebas de rendimiento se utiliza la herramienta JMeter con el objetivo de demostrar que el componente funcione correctamente. Si se encuentran deficiencias en los resultados, se realiza su registro con la ayuda de la herramienta GESPRO.

Objetivos de la prueba: Probar el rendimiento del sistema en condiciones inauditas (estresar el sistema).

Técnica que se aplica: Se aplican las pruebas de rendimiento: carga y estrés.

Criterio de aceptación: El sistema debe cumplir con los requerimientos especificados por el cliente.

Responsables de ejecutar la prueba: Probador.

Herramienta para automatización de la prueba: Jmeter.
Fecha de inicio de la prueba: (Fecha de inicio de la prueba).
Fecha de fin de la prueba: (Fecha de fin de la prueba).

2.2.5.2. Aplicación de las pruebas de seguridad

En esta prueba se intentará penetrar la seguridad del sistema mediante aplicaciones de auditoría y seguridad informática según el sistema que se esté verificando. Para la realización de esta prueba se utilizarán las herramientas brindadas por Kali Linux y las funcionalidades que provee el plugins de Firefox Web Security. En caso de lograr el objetivo de la prueba el probador registrará la no conformidad haciendo uso de la herramienta GESPRO, en caso contrario se dará la prueba por concluida, quedando demostrado que el sistema tiene bien implementada su seguridad.

Objetivos de la prueba: Penetrar la seguridad del sistema.
Técnica que se aplica: Se aplicará la técnica de fuerza bruta.
Criterio de aceptación: (Criterio para dar la prueba por satisfecha).
Responsables de ejecutar la prueba: Probador.
Herramienta para automatización de la prueba: Web Security y Kali Linux.
Fecha de inicio de la prueba: (Fecha de inicio de la prueba).
Fecha de fin de la prueba: (Fecha de fin de la prueba).

2.3. Roles y responsabilidades

Cada metodología define sus propios roles, variando la denominación y las responsabilidades de estos de una a otra. La definición de roles y responsabilidades que se ofrece a continuación está basada en los roles definidos por la metodología AUP – UCI, aunque la estrategia podrá ser adaptable a cualquier metodología de desarrollo. En caso de utilizar la estrategia de pruebas sobre otra metodología, se ha de ajustar la delegación de responsabilidades, en dependencia de las competencias que presentan los roles definidos en la misma. A continuación se describen las competencias, actividades y responsabilidades de los roles definidos en la metodología AUP-UCI que interactúan en la estrategia de pruebas:

Analista

El analista debe tener total dominio de las características del componente. Identificar y delimitar las pruebas que se requieren, así como monitorear el progreso de las mismas y los resultados en cada ciclo de prueba.

Debe tener conocimiento de cómo aplicar listas de chequeo a los artefactos generados durante el ciclo de vida de los productos. En la Fig. 14 Actividades y responsabilidades del Analista, se muestran las actividades que realiza y los artefactos por los cuales es responsable. A continuación muestran las habilidades que debe presentar un analista para desempeñar su función dentro de la estrategia.

- Capacidad de redacción y concreción.
- Conocimiento de metodologías de desarrollo de software.
- Conocimiento de ingeniería de software.

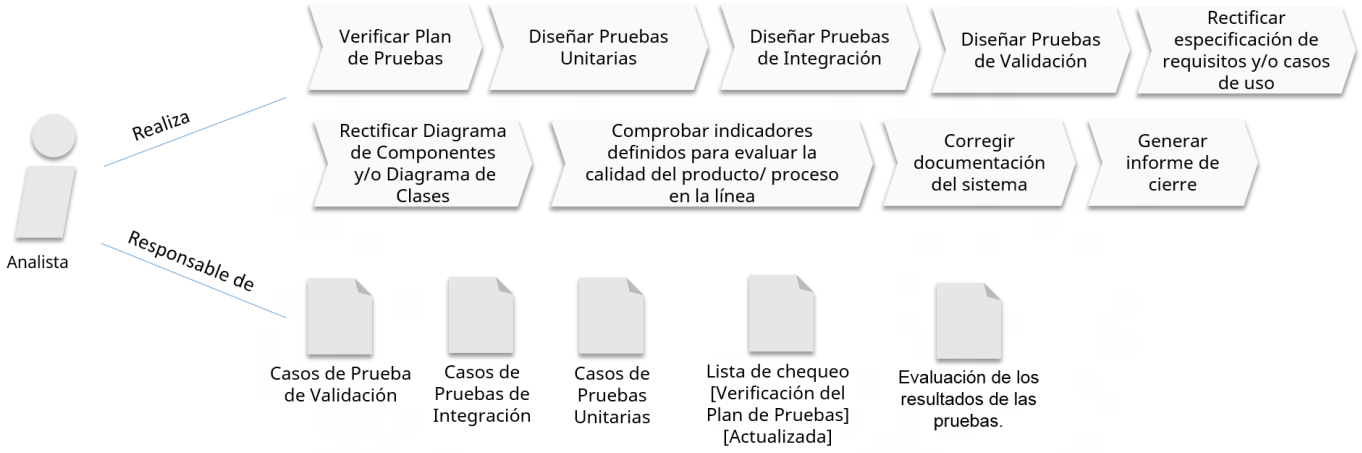


Fig. 14 *Actividades y responsabilidades del Analista. Fuente: Elaboración propia.*

Arquitecto de Software

El arquitecto es el encargado de definir todos los elementos bases de la arquitectura del proyecto. En conjunto con el desarrollador es el encargado del diseño y aplicación de las pruebas de integración. Es el responsable de la integración de los componentes del sistema. En la Fig. 15 Actividades y responsabilidades del Arquitecto de Software, se muestran las actividades que realiza y los artefactos por los cuales es responsable. A continuación muestran las habilidades que debe presentar un arquitecto de software para desempeñar su función dentro de la estrategia.

- Dominio de patrones arquitectónicos.
- Dominio de patrones de diseño.
- Dominio de herramientas de desarrollo y CASE.

- Dominio de medición de rendimiento de aplicaciones y tunning de aplicaciones (afinar la configuración para optimizar su rendimiento).
- Dominio de Estándares de información, código y comunicaciones.
- Conocimientos sobre la tecnología utilizada en el desarrollo.

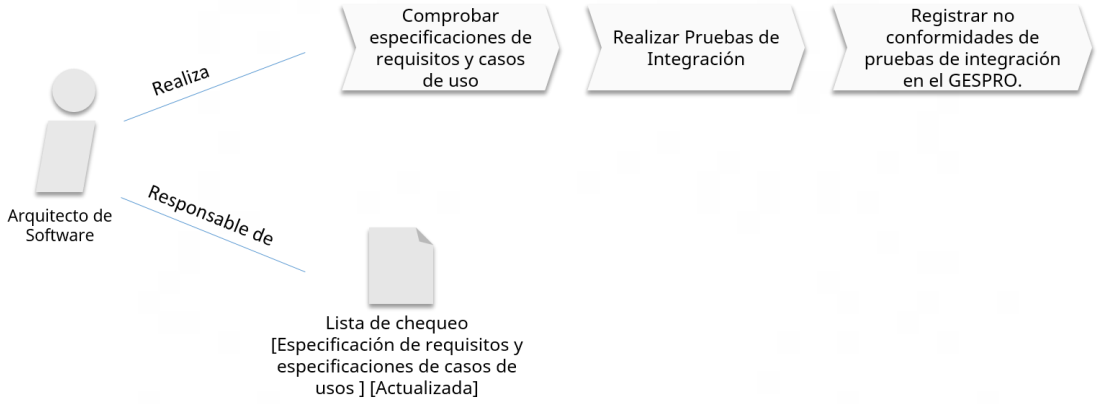


Fig. 15 *Actividades y responsabilidades del Arquitecto de Software. Fuente: Elaboración propia.*

Cliente

El cliente es el usuario final del componente. Este tiene la labor de realizar las pruebas de validación en conjunto con el desarrollador, con el objetivo de encontrar errores en el mismo. En la Fig. 16 Actividades y responsabilidades del cliente, se muestran las actividades que realiza y los artefactos por los cuales es responsable. A continuación se muestran las habilidades que debe poseer un cliente para desempeñar su función dentro de la estrategia.

- Capacidad de decisión ante un problema.
- Conocimientos de los requerimientos del componente.

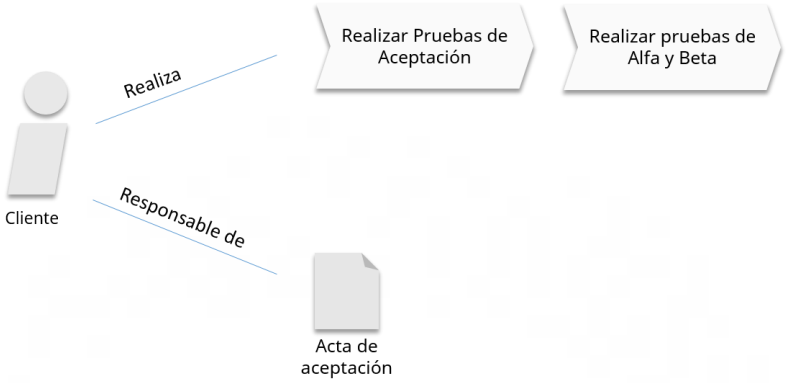


Fig. 16 *Actividades y responsabilidades del cliente Fuente: Elaboración propia.*

Desarrollador

El desarrollador elabora las pruebas de unidad, así como las pruebas de integración en conjunto con el arquitecto. Integra los componentes que forman parte de la solución. Ejecuta los casos de prueba y genera no conformidades asociadas al mismo. Registra y analiza los resultados de las pruebas. En la Fig. 17 Actividades y responsabilidades del desarrollador, se muestran las actividades que realiza y los artefactos por los cuales es responsable. A continuación se muestran las habilidades que debe presentar un desarrollador para desempeñar su función dentro de la estrategia.

- Conocimientos de técnicas de programación.
- Conocimientos de análisis y diseño de sistemas.
- Habilidad sobre lógica y algoritmos.
- Conocimientos en procesamiento de datos.
- Aptitud para identificar la mejor alternativa de solución.

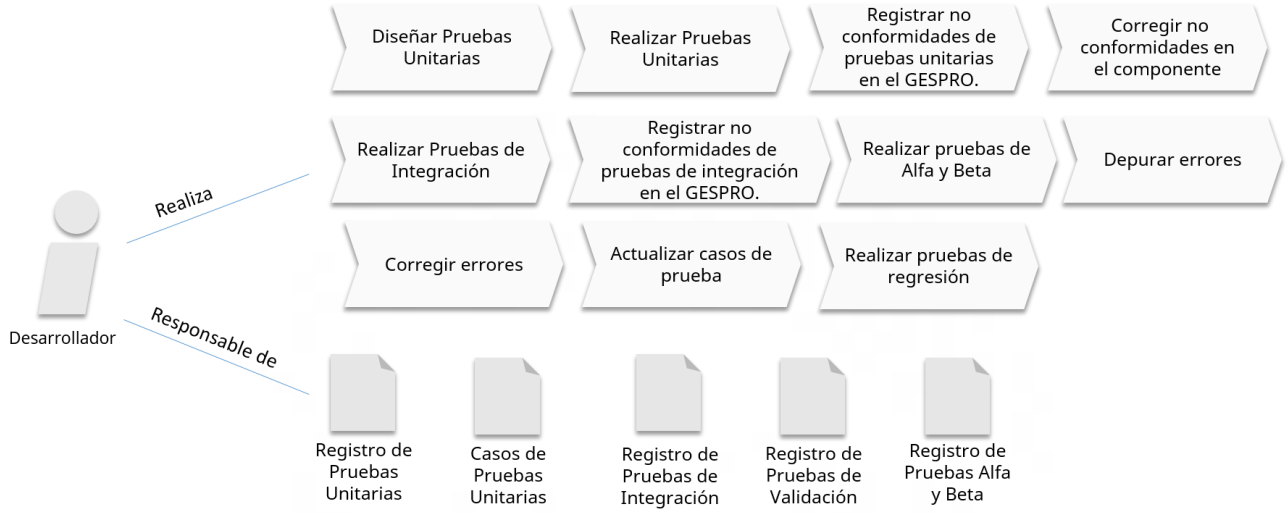


Fig. 17 *Actividades y responsabilidades del desarrollador. Fuente: Elaboración propia.*

Probador

El probador es responsable de ejecutar las pruebas como fueron planificadas, para esto debe tener conocimientos elementales de la metodología con que se trabaja, de las pruebas de software y debe ser un especialista en las herramientas escogidas para la ejecución automática de las pruebas. En la Fig. 18 Actividades y responsabilidades del Probador, se muestran las actividades que realiza y artefactos por los cuales es responsable. A continuación muestran las habilidades que debe presentar un probador para desempeñar su función dentro de la estrategia.

- Habilidad en la lectura de planes y casos de prueba.
- Conocimiento de los enfoques y técnicas de las pruebas.
- Habilidades de diagnóstico y resolución de problemas.
- Conocimiento del sistema o aplicación que se somete a prueba.
- Conocimiento de la arquitectura de red y del sistema.
- Experiencia en la utilización de herramientas de automatización de prueba.
- Conocimientos de técnicas de programación.
- Habilidades de depuración y diagnóstico de errores.

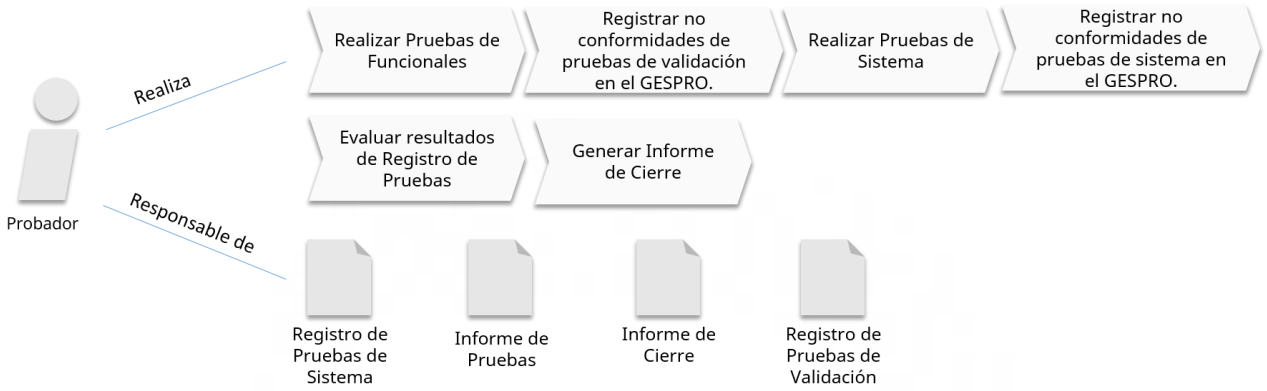


Fig. 18 *Actividades y responsabilidades del Probador. Fuente: Elaboración propia.*

Administrador de la Calidad

El administrador de calidad responde por el trabajo del equipo, auxilia con el cumplimiento del plan de pruebas y la correcta realización de la estrategia de pruebas. Su objetivo principal es asegurarse que la aplicación concuerda con los requerimientos del componente y detectar la mayor cantidad de errores. Evalúa los efectos que se consiguen al perpetrar las pruebas de calidad. Está autorizado para determinar las pruebas que se ejecutarán y verificar su aplicación. En la Fig. 19 Actividades y responsabilidades del Administrador de la calidad, se muestran las actividades que realiza y los artefactos por los que se hace responsable. A continuación se muestra las habilidades que debe presentar un administrador de la calidad para desempeñar su función dentro de la estrategia.

- Dominar técnicas para el análisis de los resultados de las pruebas por iteraciones.
- Dominar técnicas de recolección de información.
- Dominar el ciclo de desarrollo de software.
- Dominar las materias de ingeniería y gestión de software.

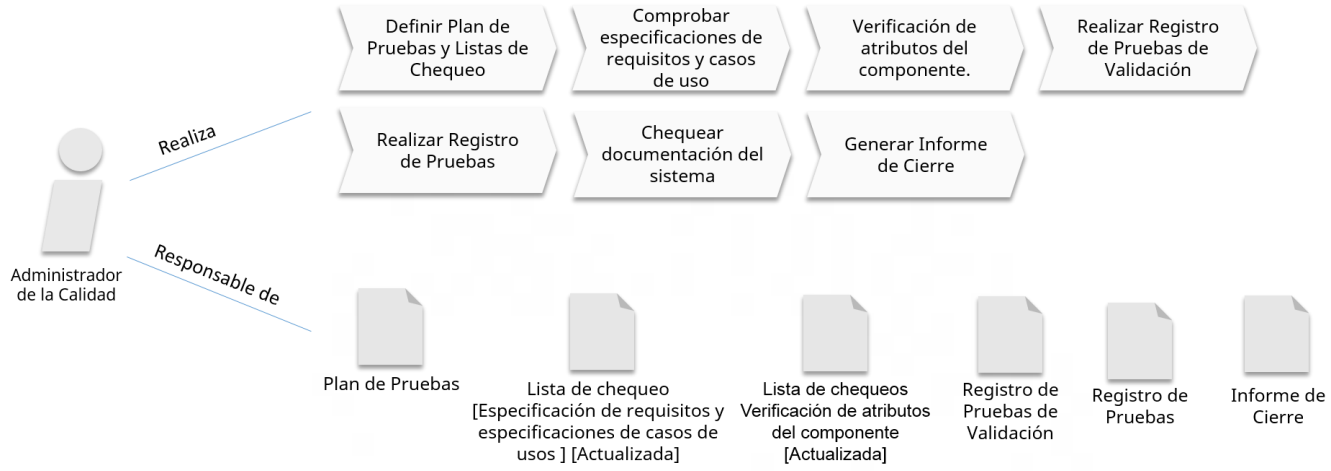


Fig. 19 *Actividades y responsabilidades del Administrador de la calidad. Fuente: Elaboración propia.*

Jefe de Proyecto

El jefe de proyecto es el encargado de liderar el equipo de trabajo del proyecto. Realiza actividades como la comprobación de los requisitos y las pruebas de aceptación. En la Fig. 20 Actividades y responsabilidades del Jefe de Proyecto, se muestran las actividades que realiza y los artefactos por los que se hace responsable. A continuación muestran las habilidades que debe presentar el jefe de proyecto para desempeñar su función dentro de la estrategia.

- Capacidad de liderazgo.
- Habilidades de comunicación y negociación.
- Conocimientos generales de las tecnologías.
- Capacidad para la toma de decisiones.
- Habilidades de trabajo en equipo.

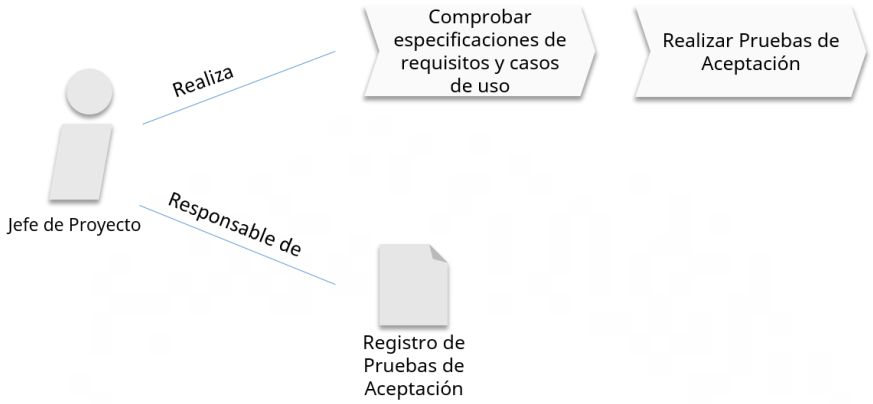


Fig. 20 *Actividades y responsabilidades del Jefe de Proyecto. Fuente: Elaboración propia.*

2.4. Descripción de los artefactos

Durante la aplicación de cada prueba que propone la estrategia se generan artefactos. A continuación se describen los artefactos generados entre los que se encuentran: plan de pruebas, casos de prueba, registro de pruebas unitarias y de integración, plan de pruebas de aceptación, lista de chequeo, registro de pruebas de seguridad, registro de pruebas de rendimiento (carga y estrés) y evaluación de las pruebas. Para obtener una descripción más detallada de cada artefacto ver expediente de la estrategia de pruebas.

- **Plan de pruebas**

En esta planilla se describe el plan de pruebas realizado por el administrador de la calidad. En este documento se definen las pruebas a utilizar en la aplicación de la estrategia, así como los responsables de cada actividad y criterios de evaluación de los resultados de las mismas.

- **Casos de pruebas**

En la estrategia se obtienen 3 tipos de casos de pruebas distintos: para pruebas unitarias, de integración y de validación, de los casos de pruebas se almacenan datos como descripción de la prueba, condiciones de ejecución y evaluación de las mismas.

- **Registro de pruebas unitarias e integración**

Este artefacto es una plantilla donde se almacenan los datos relacionados con la aplicación de las pruebas unitarias y de integración. Algunos de los datos que se recogen son el registro de pruebas y los criterios de aceptación de cada una de ellas.

- **Registro de pruebas de Aceptación**

En este artefacto se almacenan diversos datos de las pruebas aplicadas entre los que se encuentran estrategia de pruebas de aceptación, documentos generados durante la aplicación de las pruebas, cronograma de trabajo y evaluación de las pruebas.

- **Acta de Aceptación**

En este artefacto se almacenan diversos datos de la prueba aplicada entre los que se encuentran los entregables del componente, de no estar de acuerdo el cliente con algunas de la especificaciones de los entregables deja por escrito las condiciones bajo las que acepta el componente.

- **Registro de pruebas alfa y beta**

En este artefacto se almacenan diversos datos de la prueba aplicada entre los que se encuentran estrategia de pruebas alfa y beta, documentos generados durante la realización de las pruebas, cronograma de trabajo y evaluación de las pruebas.

- **Registro de pruebas de sistema**

En este artefacto se almacenan los datos relacionados con las pruebas realizadas al componente en caso de aplicar pruebas de carga y estrés, se almacenarán diversos datos de las pruebas aplicadas entre los que se encuentran estrategia de pruebas de carga y estrés, documentos generados durante la realización de las pruebas, cronograma de trabajo, recursos y evaluación de las pruebas. En caso de aplicar pruebas de seguridad se almacenarán diversos datos de la prueba aplicada entre los que se encuentran la estrategia de pruebas de seguridad, documentos generados durante la realización de las pruebas, cronograma de trabajo, recursos y evaluación de las pruebas.

2.1.8. Registro de pruebas de validación

En este artefacto se recogen los registros de todas las pruebas de validación que se hayan aplicado ya sean funcionales, alfa y beta o de aceptación, para su posterior análisis.

2.1.9. Registro de pruebas

En este artefacto se almacenan los registros de las pruebas aplicadas en la fase de ejecución, dicho artefacto pasa para la fase de evaluación de los resultados de las pruebas para su posterior análisis.

- **Listas de chequeo generadas**

Las listas de chequeo que se ejecutan se almacenan en tablas que presentan el formato que se muestra en la Tabla# 2 Lista de Chequeo, dicha especificación se encuentra registrada en el expediente de la estrategia de pruebas.

Tabla# 2 Lista de Chequeo

Elemento	Pregunta	Respuesta		Descripción
		Si	No	

2.1.10. Informe de pruebas

Este documento se redacta después de finalizar los niveles de pruebas, en el cual el probador comparará el resultado obtenido con el resultado esperado, en el cual se llega a conclusiones en conjunto con el resto del equipo de pruebas. En caso de realizar otra iteración del proceso de pruebas, llegado el fin de este, se actualizará el artefacto con los datos capturados en la iteración.

2.1.11. Informe de cierre de pruebas

Este documento se redacta al culminar la aplicación de la estrategia de pruebas. El mismo es un registro de las actividades realizadas y los resultados obtenidos a lo largo de la ejecución de la estrategia. El administrador de la calidad en conjunto con el probador serán los responsables de la elaboración del artefacto. El objetivo principal del mismo es ofrecer un registro del proceso realizado durante la aplicación de la estrategia, para poder realizar acciones preventivas que faciliten futuras mejoras del proceso. Este artefacto está conformado por diversos datos entre los que se encuentran escenarios de pruebas, evaluación de las pruebas y matriz de trazabilidad.

2.5. Conclusiones parciales

En el desarrollo de este capítulo, quedó definida la estrategia de pruebas de software a seguir para líneas de componente de software. Estableciendo una guía que garantiza la organización del trabajo, ganando en tiempo y logrando que el producto final tenga un número reducido de defectos, favoreciendo el aseguramiento de la calidad del mismo.

Capítulo 3: Validación de la estrategia de pruebas

Un aspecto fundamental de toda investigación es la validación de los resultados obtenidos en la misma. Para demostrar la veracidad de una investigación teórica se usan métodos cualitativos de pronósticos y comprobación o métodos de consultas a expertos. Estos se basan en la deducción a partir de datos empíricos y se fundamentan en la experiencia y conocimientos de un grupo de expertos poseedores de conocimientos elevados en la materia abordada (Fernández, 2014).

Los análisis cualitativos son cada vez más importantes y empiezan a formar parte de las investigaciones en diversas esferas. Uno de los métodos de pronóstico cualitativo más populares es el método Delphi. La validación de la estrategia de pruebas para líneas de desarrollo de compontes se realizará mediante este método. En este capítulo se plantean los criterios utilizados para la selección de los expertos, los cuestionarios utilizados y se estudian las respuestas manifestadas por los encuestados, arribando así a conclusiones en base al criterio generalizado de estos sobre la estrategia.

3.1. Método Delphi

Delphi es un método prospectivo de obtención de información cualitativa o subjetiva, pero relativamente precisa en contextos de información imperfecta, fruto de combinar el conocimiento y experiencia de expertos, de una forma que tiende hacia el consenso de opiniones sobre aspectos específicos, cuantificando estadísticamente, a su vez, estas opiniones.

El método Delphi consiste en la selección de un grupo de expertos a los que se les consulta su opinión sobre una determinada cuestión. La interrogación a expertos se realiza con la ayuda de cuestionarios, a fin de poner de manifiesto convergencias de opiniones y deducir eventuales consensos. La encuesta se lleva a cabo de una manera anónima. El proceso de creación del cuestionario y la selección de los expertos a participar en el estudio dependerán de los objetivos trazados para el mismo (Astigarraga, 2005).

A continuación se describen los pasos que se llevarán a cabo para garantizar la calidad de los resultados, para realizar y analizar el método Delphi:

- **Formulación del problema:** En un método de expertos, es importante definir con precisión el campo de investigación, ya que es preciso tener certeza de que los expertos consultados poseen conocimiento sobre dicho campo. Los cuestionarios elaborados deben contener preguntas precisas, cuantificables e independientes.

- **Elección de expertos:** La selección adecuada de los expertos garantiza la correcta validación de la investigación. El término “Experto” resulta ambiguo, por lo que con independencia de los títulos o nivel, el experto será seleccionado por sus conocimientos sobre el tema consultado.
- **Elaboración y lanzamiento de los cuestionarios:** Los cuestionarios se elaboran de manera que faciliten la respuesta por parte de los encuestados. Preferentemente las respuestas habrán de poder ser cuantificadas.
- **Desarrollo práctico y explotación de resultados:** Las encuestas se realizarán de forma independiente. Se realizará una ronda de cuestionarios y de ser necesario otros de forma sucesiva con el objetivo de disminuir la dispersión de las opiniones y precisar la opinión media consensuada.

En el cuestionario que se diseñó se emplearon preguntas de escala para recoger las observaciones, donde el experto debe asignar una puntuación a cada una de las posibles opciones. Estas son las preguntas de tratamiento cuantitativo y facilidad de respuesta, donde se puede observar la homogeneidad o dispersión entre ellas. Para especificar el nivel de acuerdo o desacuerdo con cada pregunta se determinó aplicar el coeficiente de Kendall.

3.2. Elección de expertos

Un experto se define como una persona reconocida como una fuente confiable de un tema, técnica o habilidad, cuya capacidad para juzgar o decidir en forma correcta, en una materia específica es elevada. Un experto es una persona con amplio conocimiento o aptitud en un área particular del conocimiento (Ericsson, 2000).

Para llevar a cabo la selección de los expertos se tuvieron en cuenta los siguientes criterios:

- Graduado de nivel superior.
- Vinculación al desarrollo de productos informáticos.
- Conocimientos y habilidades sobre calidad y pruebas de software.
- Al menos tres años de experiencia en el (los) roles que desempeña.

También se tomaron en cuenta cualidades como la formalidad, compromiso y honestidad con el fin de asegurar la confiabilidad de las opiniones brindadas.

Para la aplicación del método Delphi no existe una pauta para determinar el número de expertos con el que se ha de llevar a cabo el proceso de validación. Es recomendado que se seleccionen entre siete y treinta expertos.

Para la realización de las encuestas se utilizó una población de 11 expertos formada por especialistas de la facultad 6, de la dirección de Calidad de Software UCI y de la Dirección General de Producción. Con el objetivo de validar la estrategia desde diferentes perspectivas el panel quedó conformado por expertos que han desempeñado diferentes roles, como se puede apreciar en la Fig. 21 Roles desempeñados por panel de expertos. Además referente a la categoría científica el panel quedó conformado por ocho ingenieros en ciencias Informáticas, dos másteres y un doctor en Ciencias Técnicas.

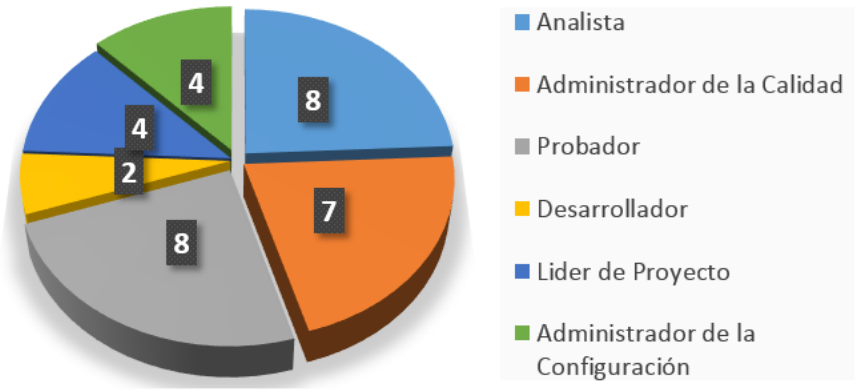


Fig. 21 *Roles desempeñados por panel de expertos. Fuente: Elaboración propia.*

A todos los expertos se les entregó un resumen de la estrategia de pruebas adjunto a dos cuestionarios, un primer cuestionario permitió la elección de 10 de ellos a partir de su nivel de competencia.

3.3. Elaboración y lanzamiento de cuestionarios

Para la elaboración de los cuestionarios primeramente se eligen los puntos o preguntas. Los puntos que componen el cuestionario deben ser precisos, cuantificables e independientes. Se ha de determinar también el tipo de respuesta que se solicitará, lo que determina el tipo de procesamiento estadístico. Para la consulta de los cuestionario se puede emplear el correo electrónico, también se envía por fax o por correo convencional con la inclusión del sobre. Es posible la entrega directa, cuando el tamaño del panel y su cercanía lo permiten.

Para la elaboración del cuestionario se tuvo en cuenta los temas principales abordados durante la investigación, con el objetivo de analizar el criterio de los expertos sobre cada uno de ellos. Los niveles, técnicas y métodos de prueba, responsabilidades por roles, y artefactos utilizados en la estrategia, se evalúan mediante preguntas independientes, con el objetivo de permitir a los expertos profundizar en sus evaluaciones. El cuestionario incluirá a su vez preguntas para que el experto autoevalúe su nivel de competencia. El cuestionario fue creado de forma tal que las respuestas fueran categorizadas en totalmente en desacuerdo (1), en desacuerdo (2), neutral (3), de acuerdo (4) y totalmente de acuerdo (5) (ver anexo #1).

3.4. Desarrollo práctico y explotación de resultados.

Para el desarrollo del método es importante obtener el nivel de competencia de los expertos seleccionados para formar parte del panel. Este nivel puede obtenerse a través de autoevaluaciones realizadas por los expertos. Para el desarrollo práctico del método Delphi también es importante definir la cantidad de rondas de consultas a efectuar. Se efectúan todas las que sean necesarias hasta llegar al consenso. Los cuestionarios se aplican de forma anónima para obtener respuestas individuales. El número de rondas es otro asunto crucial, no debe precipitarse el final del proceso realizando pocas rondas o pretender un acuerdo perfecto a través de muchas, pues puede agotar al panel y provocar el abandono de participantes. Decidir desde el inicio la cantidad de rondas que se van a realizar compromete la calidad del estudio, ello es una decisión metodológica que debe ser tomada en el proceso en dependencia de si convergen o no las opiniones de los expertos.

Al final de cada ronda se procesan las respuestas de los cuestionarios, los principales análisis estadísticos que se emplean son: media, mediana, moda, máximo, mínimo y desviación típica. En correspondencia al tipo de preguntas que se realicen, en el procesamiento de las respuestas pueden utilizarse análisis como el coeficiente de concordancia de Kendall o el coeficiente alfa de Cronbach. El resultado de dicho análisis ofrece el nivel de concordancia existente entre las opiniones de los expertos. El nivel de concordancia se utilizará como referencia para la realización de las iteraciones del método, ya que a partir de la existencia de un consenso entre los criterios se determinará si es necesaria la realización de una nueva iteración del método. Estas iteraciones se basan en la retroalimentación y repetividad con el objetivo de disminuir la dispersión de las opiniones y llegar a un consenso lo antes posible.

Coeficiente de competencia

Para la aplicación del método Delphi se propone como primer paso analizar el coeficiente de competencia del conjunto de los expertos seleccionados, con el fin de calificar de forma numérica la confiabilidad brindada por sus criterios. Este coeficiente se determina a través de la suma del coeficiente de conocimientos, el cual autoevalúa el experto con relación a su nivel de conocimientos sobre el tema abordado y el coeficiente de argumentación del cual adquirió sus conocimientos.

Para la determinación de los coeficientes de conocimiento y coeficientes de argumentación de los expertos se elaboró y aplicó un cuestionario (ver anexo # 1). Al obtener los resultados de los cuestionarios aplicados al panel de experto, se procede a calcular el coeficiente de competencia de cada experto (ver anexo # 2).

Una vez calculado el coeficiente de competencia de cada experto se procede a determinar el nivel de dicho coeficiente. Para ello se tiene en cuenta el siguiente intervalo.

- Si el coeficiente de competencia se encuentra entre 0,8 y 1,0 el coeficiente de competencia es alto.
- Si el coeficiente de competencia se encuentra entre 0,6 y 0,8 el coeficiente de competencia es medio.
- Si el coeficiente de competencia es menor que 0,6 el coeficiente de competencia es bajo.

Los expertos seleccionados para formar parte del panel de validación han de ser aquellos que sus resultados arrojen un coeficiente de competencia alto o medio. De los once expertos a los que se les aplicó el cuestionario, cuatro arrojaron un coeficiente de competencia alto y seis medio, por lo que fueron seleccionados para continuar con la ejecución del método. A continuación se muestran los resultados de dichos cuestionarios:

Tabla# 3 Nivel de competencia de expertos.

Experto	Coeficiente de competencia	Nivel de Competencia
E1	0,8	Alto
E2	0.745	Medio
E3	0.785	Medio
E4	0.69	Medio
E5	0.56	Bajo
E6	0.895	Alto
E7	0.75	Medio
E8	0.725	Medio

E9	0.835	Alto
E10	0.795	Medio
E11	0.895	Alto

Establecimiento de la concordancia de los expertos mediante el uso del coeficiente de Kendall

En busca de que la propuesta tenga mayor validez es necesario que exista un adecuado acuerdo entre las opiniones del panel de expertos, para esto se realiza una comprobación mediante el cálculo del coeficiente de concordancia de Kendall, el cual cuantifica el nivel de coincidencia de las evaluaciones arrojadas por los expertos.

Para determinar este coeficiente se desarrolla una serie de operaciones matemáticas y estadísticas, con el propósito de calcular Chi cuadrado (X^2) real, para verificar si existe o no concordancia entre los expertos (ver anexo # 3). El resultado obtenido se compara con el de la tabla Chi cuadrado (ver anexo # 4), donde si Chi cuadrado real es menor que Chi cuadrado entonces existe una concordancia entre el panel de expertos. Como resultado de esta comparación se obtuvo que 0.3645< 20.0902, por lo que se concluye que existe concordancia entre los criterios de los expertos que conforman el panel y es válido continuar con el procesamiento de los cuestionarios.

Procesamiento del Resultado del Cuestionario

A partir del procedimiento realizado para la explotación de los resultados (ver anexo # 4) se obtiene el grado de adecuación de las preguntas sometidas a la valoración de los expertos. En la siguiente tabla se presentan la categorización de los diferentes aspectos abordados en los cuestionarios realizados.

Tabla# 4 Categorización de los aspectos.

Pregunta	Grado de adecuación
P1	De acuerdo
P2	De acuerdo
P3	De acuerdo
P4	De acuerdo
P5	De acuerdo
P6	De acuerdo
P7	De acuerdo

P8	De acuerdo
P9	Neutral

A partir de los resultados reflejados en la tabla anterior se puede concluir que el panel de expertos está de acuerdo con lo definido en la investigación, ya que en la gran mayoría de los aspectos a evaluar coincide el grado de adecuación. A continuación se representa de forma gráfica los resultados obtenidos.

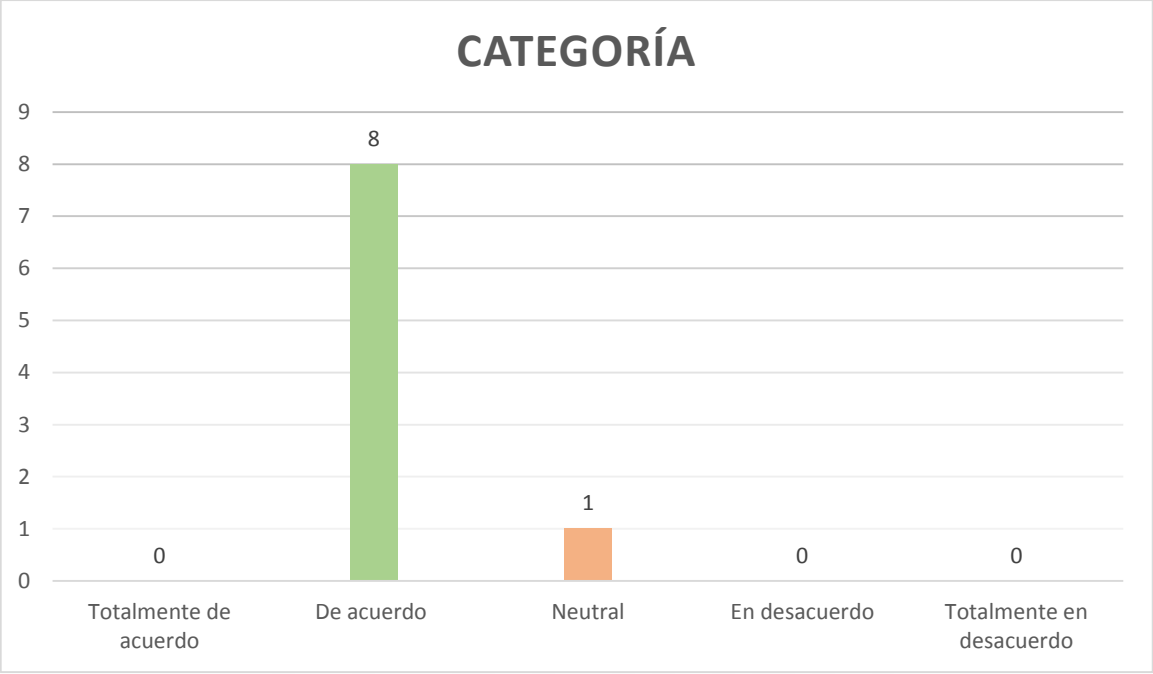


Fig. 22 *Categoría de los resultados de la encuesta. Fuente: Elaboración propia.*

3.5. Conclusiones parciales

Los resultados anteriores mostrados en la Fig. 22 Categoría de los resultados de la encuesta, demuestran la validez de la estrategia propuesta para ser utilizada en la UCI, pues según el criterio de los expertos la misma contribuye a establecer buenas prácticas para el desarrollo basado en componentes. Aumentar la calidad y el nivel de reutilización de los componentes desarrollados. Disminuir los problemas en la integración de productos y que se garantice la medición y mejora de los indicadores de calidad en líneas de componentes.

Conclusiones Generales

- El diseño de la estrategia de pruebas para líneas de desarrollo de componentes en la Universidad de las Ciencias Informáticas constituye una guía para el proceso de desarrollo a más bajo nivel y permite la validación interna de los componentes a nivel de línea.
- Los procesos identificados, las actividades correspondientes en el desarrollo de la estrategia y la concepción iterativa permiten el cubrimiento de las fases previstas y la mejora continua de la calidad de los componentes desarrollados.
- Se identificaron las pruebas en diferentes niveles, métodos y técnicas acordes a las características de los componentes para verificar la calidad de acuerdo a los requisitos definidos.
- La aplicación del criterio de expertos demuestra consenso en el nivel de aceptación de la estrategia diseñada y valida la propuesta para ser aplicada en el aseguramiento de la calidad en líneas de desarrollo de componentes de software en la UCI.

Recomendaciones

Al término de la investigación se considera recomendable para trabajos futuros:

- Diseñar actividades para la automatización de procesos de verificación de la calidad de los componentes en actividades de la estrategia como la generación de casos de pruebas, la ejecución de pruebas y las actividades relacionadas a la obtención de informes de resultados de pruebas.
- Definir mecanismos para garantizar la medición efectiva de indicadores propios de las líneas de desarrollo de componentes de software y los definidos en el área de procesos de Medición y Análisis de la Universidad de las Ciencias Informáticas.

Referencias Bibliográficas

1. **adimedia.net. 2014.** Herramientas de pruebas. 2014.
2. **Astigarraga, Eneko. 2005.** EL MÉTODO DELPHI. 2005.
3. **Campos, Miguel Ángel Canela. 2002.** Gestión De La Calidad . 2002.
4. **Casallas, Rubby. 2005.** Mejoramiento del Proceso de Pruebas en un Contexto de Desarrollode Software Globalizado. 2005.
5. **Chicago, Loyola University. 2007.** Libro:Test Strategy Document. 2007.
6. **CMMI. 2010.** Mejora de los procesos para el desarrollo de mejores productos y. s.l. : Editorial Universitaria Ramón Areces, 2010. Vol. V 1.3.
7. **CZARNECKI. 2006.** Feature models are views on ontologies. 2006. 2006 Software Product Line Conference.
8. **Deris, Alejandro Monteverde Hill. 2005.** Pruebas De Soportabilidad . 2005.
9. **Ericsson, K. Anders. 2000.** Expert Performance and Deliberate Practice. 2000.
10. **Española, Real Academia. 2015.** Diccionario de la Real Academia Española. [En línea] 2015. [Citado el: 20 de 4 de 2015.] http://www.rae.es.
11. **Espinoza, Vanessa Quinta Bárbara. 2005.** 'Pruebas De Rendimiento. 2005.
12. **Fernández, Sandra Hurtado de Mendoza. 2014.** Sistema Automatizado Del Método De Consultas a Expertos. 2014.
13. **Figueredo, Clara Elena Brizuela. 2013.** Datamart for the Maintenance Company of Electrogene Groups. 2013.
14. **Hammond, David. 2015.** Web Development Tools. [En línea] 2015. [Citado el: 24 de 3 de 2015.] http://www.usefulusability.com/24-usability-testing-tools/.
15. **Huerta, Rosendo. 2006.** Proceso De Análisi Integral De Disponibilidad Y Confiabilidad Como Soporte Para El Mejoramiento Continuo De Las Empresas. 2006.
16. **IEEE, Standard. 1990.** Compilation of Ieee Standard Computer Glossaries. [En línea] 1990. [Citado el: 12 de 3 de 2015.]
17. **ISO:9126-1. 2000.** Factores de Calidad del Software. [En línea] 2000. [Citado el: 7 de 6 de 2015.] www.iso.org.
18. **Linstone, Harold A. 2002.** The Delphi Method. 2002.

19. **Loos, Peter. 2002.** Specification of Business Components. Objects, Components, Architectures, Services and Aplications for a Networked World. . 2002.

20. **Martínez, Jesús Badenas. 2011.** Incorporación de pruebas automáticas sobre transformaciones de modelos en el entorno de integración continua de MOSKitt. 2011.

21. **Montilva, Jonás A. 2002.** Desarrollo De Software Basado En Componentes. 2002.

22. **Pressman, Roger. 2005.** Ingenería de Software. Un enfoque práctico. 2005.

23. **Rojas, Maribel Ariza. 2004.** Introducción Y Principios Básicos Del Desarrollo De Software Basado En Componentes. 2004.

24. **Sommerville, Ian. 2007.** Software Engineering. 2007.

25. **Standard, IEEE. 1990.** Compilation of IEEE Standard Computer Glossaries. [ed.] Institute of Electrical and Electronics Engineers Computer dictionary. 1990.

26. **Szyperski, Clemens. 2002.** Component software: Beyond object-oriented programming. 2da Edicion. Reading, Massachusetts : Addison-Wesley , 2002.

27. **Tenorio, Roberto Ruiz. 2010.** Las pruebas de software y su importancia en las organizaciones. 2010.

28. **UCI. 2015.** tp://mejoras.prod.uci.cu/. 2015.

29. **Vázquez, Roberto Hugo. 2006.** Taller de Calidad de Software: Introducción a la Calidad de Software. 2006.

30. **Wesley, Addison. 1999.** Software Architecture in Practice. 6ta Edicion. Reading, Masachusetts : Addison Wesley, 1999.

31. **White, Stephen y Miers, Derek. 2009.** Guia de Referencia y Modelado BPMN. 2009.

32. **www.kali.org. 2015.** Kali Linux | Penetration Testing and Ethical Hacking Linux. [En línea] 2015. [Citado el: 25 de 2 de 2015.] www.kali.org.

33. **XEDRO. 2013.** Metodología UCI. 2013.

Bibliografía

1. **Astigarraga, Eneko. 2005.** EL MÉTODO DELPHI. 2005.
2. **CZARNECKI. 2006.** Feature models are views on ontologies. 2006. 2006 Software Product Line Conference.
3. **Linstone, Harold A. 2002.** The Delphi Method. 2002.
4. **Pressman, Roger. 2005.** Ingenería de Software. Un enfoque práctico. 2005.
5. **Sommerville, Ian. 2007.** Software Engineering. 2007.
6. **Standard, IEEE. 1990.** Compilation of IEEE Standard Computer Glossaries. [ed.] Institute of Electrical and Electronics Engineers Computer dictionary. 1990.
7. **Tenorio, Roberto Ruiz. 2010.** Las pruebas de software y su importancia en las organizaciones. 2010.
8. **UCI. 2015.** tp://mejoras.prod.uci.cu/. 2015.
9. **Vázquez, Roberto Hugo. 2006.** Taller de Calidad de Software: Introducción a la Calidad de Software. 2006.
10. **White, Stephen y Miers, Derek. 2009.** Guia de Referencia y Modelado BPMN. 2009.
11. **XEDRO. 2013.** Metodología UCI. 2013.