Universidad de las Ciencias Informáticas

Facultad 5

Módulo de apoyo para la generación de productos de trabajo en la disciplina de Pruebas

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

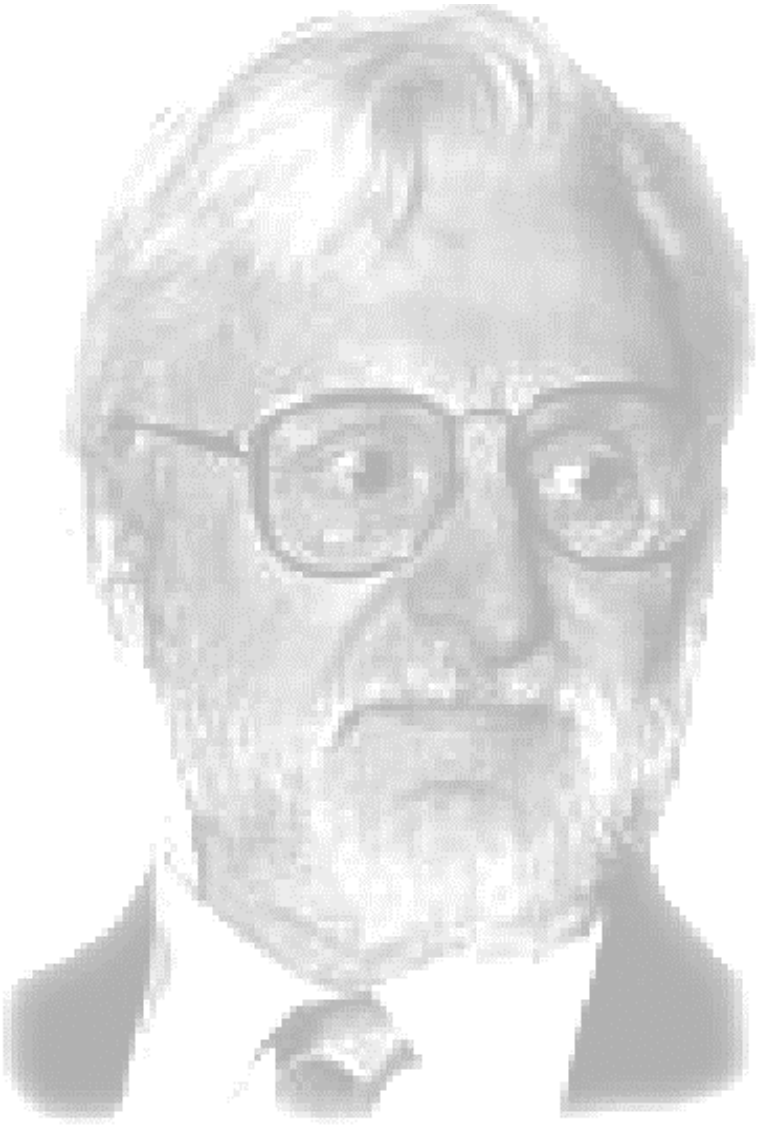
Autor (a): Dianne Cordero García

Tutor (es): Ing. Andy Hernández Paez

Co-tutor (es): Ing. Ernesto Gutierrez Ramos

Ing. Yanet L. Garbey Gainza

La Habana, Cuba
Junio 2016



"Calidad es una característica de fortaleza y estabilidad que es reconocida por un proceso inerte. Debido a que las definiciones son producto de un pensamiento formal y rígido, la calidad no puede ser definida."

Robert M. Pirsing.

Declaración de autoría

Declaro por este medio que yo Dianne Cordero García, soy la autora principal de este trabajo final de tesis “Módulo de apoyo para la generación de productos de trabajo en la disciplina de Pruebas” y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Dianne Cordero García

Autor

_____

Ing. Andy Hernández Paez

Tutor

_____

Ing. Ernesto Gutierrez Ramos

Co-tutor

_____

Ing. Yanet L. Garbey Gainza

Co-tutor (a)

Datos de contacto

Tutor: Ing. Andy Hernández Paez.

Edad: 27.

Ciudadanía: cubana.

Institución: Universidad de las Ciencias Informáticas (UCI).

Títulos: Ingeniero en Ciencias Informáticas.

Síntesis del Tutor: Graduado de Ingeniería en Ciencias Informáticas en junio del 2012, en la Universidad de Ciencias Informáticas. Profesor Instructor de la disciplina Ingeniería y Gestión de Software. Actualmente profesor del centro Vertex Entornos Interactivos 3D. Posee 5 años de experiencia como Analista de sistema en proyectos informáticos de Informática Industrial y Realidad Virtual. Se desempeña como Asesor de Calidad de software del Centro Vertex y dirige el grupo de Ingeniería y Calidad de Software de dicho centro. Especialista en el marco de trabajo de la Ingeniería de Requisitos y Arquitectura de la Información. Presenta varias publicaciones y participación en eventos.

E-mail: andyhp@uci.cu

Co-tutor: Ing. Ernesto Gutierrez Ramos.

Edad: 24.

Ciudadanía: cubana.

Institución: Universidad de las Ciencias Informáticas (UCI).

Títulos: Ingeniero en Ciencias Informáticas.

Síntesis del Co-tutor: Graduado de Ingeniero en Ciencias Informáticas en la novena graduación de la Universidad de las Ciencias Informáticas (UCI). Actualmente trabaja en la línea web del Departamento de Construcción de Componentes del Centro de Entornos Interactivos 3D (VERTEX) de la UCI.

E-mail: egutierrezr@uci.cu

Co-tutor (a): Ing. Yanet Liliana Garbey Gainza

Edad: 24.

Ciudadanía: cubana.

Institución: Universidad de las Ciencias Informáticas (UCI).

Títulos: Ingeniería en Ciencias Informáticas.

Síntesis del Co-tutor: Graduada de Ingeniería en Ciencias Informáticas en junio del 2015, en la Universidad de Ciencias Informáticas. Recién Graduada en Adiestramiento prestando servicios como profesora en la disciplina Idioma Extranjero. Se encuentra en el primer año de experiencia como Analista de sistema en proyectos informáticos de Realidad Virtual. Se desempeña como Administradora de la Configuración del Centro Vertex y forma parte del grupo de Ingeniería y Calidad de Software de dicho centro. Presenta publicaciones en eventos científicos.

E-mail: liliana@uci.cu

Dedicatoria

A mi familia, en especial a mis padres, mi hermano y a todas las personas que me quieren y siempre confiaron en mí brindándome su apoyo para poder realizar este sueño, quiero regalarles este momento y honrarlos por tanto amor y dedicación. Los quiero mucho.

Agradecimientos

Después de muchos años de estudio y esfuerzo, hoy prácticamente termina mi vida como estudiante universitaria. Para hacer realidad este sueño muchas personas me ayudaron, apoyaron y depositaron su confianza en mí. Hoy tengo la oportunidad de hacerles saber cuan agradecida estoy.

A mis queridos padres por todo su esfuerzo y sacrificio, por su cariño, su apoyo incondicional, gracias por estar siempre cuando los necesité. Quiero que sepan que los amo y los admiro mucho; y este logro es de ustedes también, nunca les podré estar lo suficientemente agradecida por todo lo que han hecho por mí, gracias por ser mi guía, mi apoyo y mi vida.

A mi familia política, que desde el primer día me abrieron las puertas de su casa sin apenas conocerme, gracias por tratarme como su hija o nieta, y por permitirme formar parte de su familia. A mi compañero en la vida personal José Luis Corredera Gutiérrez, por soportarme y quererme, por apoyarme y darme fuerzas para seguir adelante en los momentos difíciles y sobre todo por aguantar todos mis caprichos.

Por otra parte, quisiera agradecerle a mi tutor y co-tutores por todo su esfuerzo y paciencia, son una de las piezas más importantes en esta tesis. A las personas que forman el día a día de mi vida junto a mi familia, mis compañeros de la universidad con los cuales he vivido grandes momentos, los aprecio muchísimo y sepan que pueden contar conmigo para lo que sea, que aquí tienen una amiga más.

En fin, a todos aquellos que de una forma u otra han formado parte de mi vida y me han apoyado, gracias.

Resumen

En el presente trabajo de diploma se realiza un estudio del arte de la disciplina de Pruebas en proyectos de desarrollo teniendo en cuenta modelos y estándares de calidad. Se investiga, además, acerca de los tipos de pruebas, clasificadas en dinámicas y estáticas. Aunque, estos tipos de pruebas son indispensables para cualquier proyecto en desarrollo, la investigación se centra en las pruebas estáticas, puesto que permiten evaluar la conformidad del producto respecto a sus especificaciones y sobre todo aseguran la integridad técnica y conceptual de los cambios que pudiesen existir sobre el mismo producto. Por otra parte, también fue necesario el estudio de los productos de trabajo contenidos en el Expediente de Proyecto de Desarrollo en su versión 4.0 que guardan una indiscutible relación con la disciplina de Pruebas. También se abarca el estudio de algunas de las funcionalidades existentes en herramientas precedentes que abarcan el proceso de dicha disciplina. Además, se realiza un análisis conciso de las tecnologías y lenguajes para el desarrollo de la propuesta de solución informática, así como el patrón arquitectónico y los patrones de diseño de los cuales se hará uso. Finalmente se valida la propuesta de solución mediante la implementación de un módulo de apoyo para la generación de productos de trabajo en la disciplina de Pruebas, contribuyendo de esta manera con una mejora considerable en cuanto a la calidad y agilización de la generación de dichos productos de trabajo.

Palabras clave: disciplina de Pruebas, Expediente de Proyectos de Desarrollo 4.0, productos de trabajo.

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

El hombre en su afán de desarrollo persigue mejorar su calidad de vida, con tal fin no sólo se ha centrado en crear innovaciones que le aporten beneficios, también mediante esta forma de evolución van aparejado los procesos de calidad seguidos para el avance tecnológico. La industria del software, a pesar de ser muy joven ha adoptado con gran empeño los principios de calidad indispensables para el desarrollo de sus productos. Una explicación a esta realidad se fundamenta en la amplia competencia que existe a nivel mundial en este campo a fin de cumplir con las expectativas del cliente.

Actualmente cualquier empresa de software que se haya propuesto como meta satisfacer a sus clientes, mantener una imagen reconocida entre sus semejantes o de una forma más lucrativa, obtener grandes ganancias, simplemente está obligada a asegurar la calidad de sus proyectos. Este es un proceso que se ha convertido en un objetivo estratégico para las organizaciones a través de la mejora continua de sus procesos a diferentes niveles de madurez y capacidad.

Los aspectos relacionados con la calidad en los proyectos de software, no sólo hacen referencia a la tecnología utilizada o a los conocimientos de las personas responsables de su desarrollo, se trata de mejorar y asegurar todos los procesos involucrados para satisfacer las expectativas del cliente al entregar un producto final estable y confiable. Este producto de software debe de encontrarse ajustado a las especificaciones funcionales que hayan sido documentadas. A esto se suman las expectativas del cliente asociadas a los riesgos, prestaciones, plazos de entrega y costos.

Para obtener mayor confianza en que el software se entregará en tiempo y trabajará como es debido, al mismo se le deben realizar diversas pruebas durante todo su ciclo de vida. Esta evaluación es especialmente rigurosa, debido a que el software es discreto, intangible, invisible e interdependiente, la misma se ha convertido en una disciplina crítica dentro de la Ingeniería de Software y es parte esencial de cualquier proceso de desarrollo.

En la actualidad las pruebas de software se han convertido en una importante herramienta para el aseguramiento de la calidad de los procesos y productos durante el proceso de desarrollo de software. El hecho de someter los sistemas a estas pruebas ayuda a reducir el riesgo de complicaciones durante las operaciones y a contribuir a la calidad del sistema, siempre y cuando los defectos detectados se corrijan antes que el sistema se ponga a disposición del usuario final para su uso operativo.

Existen diferentes tipos de pruebas de software asociadas a determinadas técnicas que establecen el grado de profundidad en el cual se diseñarán, ejecutarán y evaluarán, teniendo como referencia la estructura interna de la aplicación; las principales técnicas son: Caja Blanca, Caja Gris y Caja Negra [1].

Además de las técnicas expuestas anteriormente, existen técnicas estáticas, tales como, las inspecciones que no son más que exámenes visuales que se realizan a un software con el objetivo de detectar anomalías.

La Universidad de las Ciencias Informáticas (UCI) como parte del proceso de informatización de la sociedad cubana, ha creado un conjunto de proyectos de software los cuales están orientados hacia varias áreas de la vida cotidiana; para ello cuenta con varios centros de desarrollo de software asociados a ella. Tal es el caso del Centro de Entornos Interactivos 3D (VERTEX), el cual se especializa en la creación de productos de software sobre diversas líneas de desarrollo, entre las que se encuentran: Juegos Serios (videojuegos), Entrenadores 3D, Paseos Virtuales y Simulación Computacional 3D. En este centro se encuentra el grupo de Ingeniería y Calidad de Software, el cual se encarga de aplicar objetivamente las buenas prácticas existentes para el análisis, diseño y calidad de proyectos de software.

En el Proceso de Desarrollo de Software (PDS[1]) de las líneas de desarrollo del centro VERTEX antes mencionadas y en relación con la disciplina de Pruebas resulta indispensable la gestión de productos de trabajo (artefactos), tales como: Lista de verificación de revisiones de inconsistencias, Diseño de Casos de Prueba y Registro de defectos y dificultades detectados, contribuyendo en gran parte a ayudar a los analistas de sistemas, administradores de calidad y probadores; facilitando de esta manera la culminación y entrega del software en desarrollo.

En aras de viabilizar el trabajo en la disciplina de Pruebas en proyectos de software del centro VERTEX se analizaron algunos puntos cruciales que limitan las prácticas de la misma:

1. En ocasiones la disponibilidad, centralización y uniformidad de los productos de trabajo mencionados con anterioridad entorpecen el proceso para medir la calidad del producto final, pues los mismos constituyen productos de trabajo indispensables que tienen que ser generados en el momento adecuado para garantizar la calidad del software.
2. No existe un registro histórico confiable de las inconsistencias identificadas en los proyectos de software al finalizar cada disciplina y por consecuencia no se puede determinar una evaluación real sobre las inconsistencias detectadas a lo largo del PDS, lo que afecta además la medición sistemática de la calidad del producto.
3. Cuando se diseñan los casos de pruebas del software en muchas secciones de estos, existen repeticiones de escenarios que deben tenerse en cuenta también para otros casos de prueba, sin embargo, estos son obviados. Además, no existe una relación entre las secciones especificadas en los casos de uso del sistema y las definidas para los casos de prueba.

4. No existe relación en los tipos de variables de entradas de los casos de prueba, los definidos en la aplicación y especificados en los casos de uso del sistema, lo que afecta el registro de inconformidades del proyecto y la medición de la calidad del producto final.
5. Se desconocen las evidencias y especificaciones de los defectos y dificultades detectados en algunas líneas de desarrollo de software de proyectos del centro, tales como: Juegos Serios (videojuegos), Entrenadores 3D, Paseos Virtuales y Simulación Computacional 3D; lo que provoca descontrol en las no conformidades relacionadas con los productos de trabajo y el software.

Teniendo en cuenta la situación antes expuesta se plantea como **problema de la investigación:** ¿Cómo contribuir a la generación de los productos de trabajo en la disciplina de Prueba del Expediente de Proyecto de Desarrollo 4,0 para el centro VERTEX?

Teniendo como **objeto de estudio**: Productos de trabajo para medir la calidad, relacionados con la disciplina de Pruebas. A partir del mismo, se delimita como **campo de acción**: Productos de trabajo relacionados con la disciplina de Pruebas del Expediente de Proyecto de Desarrollo 4.0 para medir la calidad en líneas de desarrollo del centro VERTEX.

Con vista a la solución del problema científico se plantea como **objetivo general**: Desarrollar un módulo web de apoyo a la generación de productos de trabajo del Expediente de Proyectos de Desarrollo 4.0 en la disciplina de Pruebas.

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

1. Elaboración del marco teórico de la investigación a partir del estado del arte existente sobre el tema.
2. Análisis de los productos de trabajo del expediente de proyectos de desarrollo 4.0 para el área de proceso de Administración de Requisitos (REQM) del nivel 2 de CMMI, relacionados con la disciplina de Pruebas para seleccionar los artefactos a generar.
3. Selección de la metodología, herramientas y tecnologías para el desarrollo del módulo como propuesta de solución a integrarse con el sistema de gestión de Artefactos del Expediente de Proyecto (AREXPRO).
4. Generación de los artefactos que corresponden a los flujos de trabajo definidos por la metodología seleccionada como pruebas del trabajo realizado.
5. Implementación de un módulo que facilite la generación de productos de trabajo en la disciplina de Pruebas para escenarios de desarrollo de proyectos del centro VERTEX.

6. Validación del módulo mediante pruebas de funcionalidad, rendimiento, integración y aceptación, para mitigar errores en la propuesta de solución.

Para la realización de estas tareas se emplearon los métodos de investigación que se describen a continuación:

Métodos teóricos:

- Histórico-lógico: este método se utiliza para el estudio de trabajos similares e investigaciones que abordan el tema de la disciplina de Pruebas a nivel nacional e internacional, así como sistemas relacionados con esta disciplina, que sirvan como punto de partida para el desarrollo de la solución en su conjunto.
- Analítico-sintético: este método se utiliza para analizar desde diferentes aristas los conceptos asociados a la disciplina de Pruebas y sintetizar la información recopilada, permitiendo describir las características generales y las relaciones esenciales entre estas.

Métodos empíricos:

- Observación: Su empleo permitió observar el funcionamiento y evolución de herramientas existentes para la automatización de pruebas de software, en aras de reconocer las necesidades específicas y características que debe de poseer la propuesta de solución.

El presente documento está compuesto por tres capítulos:

- **Capítulo 1. Fundamentación teórica:** Se hace referencia a los principales aspectos contenidos en la disciplina de Pruebas tales como, la calidad, los diferentes tipos de pruebas que se pueden aplicar sobre un software, entre otros. Además, se explica detalladamente el proceso de prueba que se lleva a cabo en los proyectos según la línea de desarrollo en la cual están enmarcados. Por otra parte, se realiza un análisis de sobre el funcionamiento de algunas herramientas para la automatización de pruebas. Por último, se muestra en detalles el estudio realizado sobre las distintas herramientas, tecnologías y metodologías que fueron analizadas con el fin de tomar decisiones para la futura implementación del sistema propuesto.
- **Capítulo 2. Características y diseño del sistema:** se describe la solución propuesta para darle respuesta a la situación planteada. Además, se realiza una descripción detallada del sistema a través de los requisitos funcionales y no funcionales. Se realizan actividades de análisis y diseño para la construcción de los diagramas del caso de uso del sistema, de clases, de interacción, así como el modelo de dominio y modelo de datos.

- **Capítulo 3. Implementación y prueba del sistema:** se describe el desarrollo de la solución, se especifica el estilo de programación, se presenta el modelo de implementación mediante el diagrama de despliegue y de componente que resultó del diseño realizado de cada uno de los casos de uso del sistema y los casos de prueba realizados a los casos de uso más significativos.

Capítulo #1. Fundamentación teórica

1.1 Introducción

En el presente capítulo se abordan los principales conceptos que ofrecen una mejor comprensión del tema a investigar. Se explica de manera detallada como se lleva a cabo el proceso de Pruebas, del Sistema de Gestión de Calidad implantado en la UCI con el objetivo de lograr alcanzar una mayor calidad en torno al software que se libera. Además, se realiza un estudio de las principales herramientas existentes a utilizadas para la automatización de pruebas de software, con el objetivo de resumir las funcionalidades y características que poseen. Se explica la metodología utilizada, por último, se realiza una fundamentación de las tecnologías, herramientas y lenguajes que serán de uso en el desarrollo de la solución informática.

1.2 Calidad de software

La Gestión de la Calidad de Software es una actividad esencial en cualquier empresa de software para asegurar la calidad de sus productos y la competitividad frente a la oferta del mercado. Es un conjunto de actividades de la función general de la Dirección que determina la calidad, los objetivos y las responsabilidades. El propósito de la Administración de la Calidad de Software es, en primer lugar, entender las expectativas del cliente en términos de calidad, y poner en práctica un plan proactivo para satisfacer esas expectativas. Dado que la misma está definida por el cliente, podría parecer que es completamente subjetiva, de cualquier forma, hay muchas cosas acerca de la calidad que pueden hacerse objetivamente. Esto requiere examinar cada una de las características individuales del software y determinar una o más métricas que pueden recolectarse para reflejar dichas características. Por ejemplo, una característica de calidad puede ser que la solución tenga la menor cantidad de errores. Esta característica puede medirse contando los errores y defectos de la solución [2]. La Gestión o Administración de la Calidad se aplica normalmente a nivel empresa. También puede haber una gestión de la calidad dentro de la gestión de cada proyecto.

La Administración de la Calidad no es un evento, es un proceso y una forma de pensamiento. Un producto de software consistente, de alta calidad no puede producirse a partir de un proceso malo. Existe la necesidad de un ciclo constante para medir la calidad, actualizar el proceso, medir otra vez, actualizar, entre otros. Para hacer que la Administración de calidad del software funcione, es vital recolectar métricas. Si no se capturan métricas será difícil mejorar los procesos a partir de una iniciativa de Administración de calidad [2].

A partir del 2005 en la Universidad se introducen las pruebas de liberación a los productos de trabajo que constituyen entregables al cliente; y a medida que transcurre el tiempo se van presentando nuevas

conquistas con resultados bastante satisfactorios. Estos resultados fueron evolucionando a partir de aportes teóricos y de la retroalimentación de la experiencia práctica de especialistas de la Dirección y más tarde en el Centro de Calidad que existía en el área productiva de la Universidad.

En el 2012 como parte del traspaso de la UCI al Ministerio de la Educación Superior (MES) el Centro de Calidad pasa a ser el Centro Nacional de Calidad de Software del Ministerio de las Comunicaciones. Este centro ha ido brindando los servicios de pruebas de liberación, revisiones y auditorías a la Universidad, en el 2013 realizaron 119 revisiones, 13 auditorías y se liberaron 216 productos de trabajo a través de las pruebas de liberación. No obstante, existen los siguientes elementos que llevan al análisis de una propuesta:

- Elevar la calidad de los productos de trabajo Especificación de Requisitos, Documento de Arquitectura de Software y Diseño de la Base de Datos por el impacto que tiene en la solución y la sostenibilidad del producto, a través de revisiones técnicas formales por pares de expertos.
- Las aplicaciones se enfrentan a las pruebas de liberación/aceptación sin evaluarse en la Universidad, al enfrentarse por primera vez en esta etapa requieren de varias iteraciones y se detectan un número considerable de no conformidades. Por lo que es necesario verificar las soluciones antes de ser evaluado por una entidad externa, en vista a garantizar un mayor control de la calidad de las soluciones.
- Los servicios de auditorías y revisiones evalúan la adherencia a las políticas, procesos y productos de trabajo establecidos para la actividad de desarrollo producción en la Universidad, y se considera que el control y seguimiento a los procesos no deben garantizarse solamente con la evaluación de una entidad externa. Las evaluaciones de adherencia a proceso son el punto de partida para la identificación de las oportunidades de mejora.

Para la actividad de desarrollo producción la Universidad cuenta con la Red de Centro, áreas especializadas para el seguimiento y control, la calidad, el soporte y el diseño comunicacional. Cada centro dispone de un asesor de calidad y en algunos casos de grupos de trabajos para ejecutar actividades de aseguramiento y control de la calidad. Los centros productivos y los departamentos docentes de la especialidad disponen de personas que poseen experiencias en la producción y han obtenido resultados satisfactorios en su desempeño, los cuales pueden ser considerados expertos en algunos roles y en la ejecución de los procesos [3].

A continuación, se muestra una representación gráfica del Sistema de Gestión de la Calidad (SGC) propuesto y que debe ser ejecutado por cada proyecto de software UCI:

Figura 1: Representación gráfica del SGC [3]**.**

1.3 Modelos y/o Estándares de calidad

Los Modelos de Calidad son aquellos documentos que integran la mayor parte de las mejores prácticas, proponen temas de administración en los que cada organización debe hacer énfasis, integran diferentes prácticas dirigidas a los procesos clave y permiten medir los avances en calidad. Los Estándares de Calidad son aquellos que permiten definir un conjunto de criterios de desarrollo que guían la forma en que se aplica la Ingeniería del Software. Los estándares suministran los medios para que todos los procesos se realicen de la misma forma y son una guía para lograr la productividad y la calidad. El Modelo / Estándar de Calidad del Software consiste en reunir todas las actividades y funciones, de forma tal que ninguna de ellas esté subordinada a las otras y que cada una se planee, controle y ejecute de un modo formal y sistemático [4].

El uso de Modelos y Estándares de Calidad del Software ayuda a lograr una mejor Gestión de la Calidad; la cual se debe de centrar en tres niveles de trabajo: nivel de organización, nivel de proyecto y nivel de producto de software.

A continuación, se muestra una tabla resumen con un listado de los modelos/estándares de calidad del software a nivel de proceso y producto respectivamente [2].

Tabla 1: Modelos / Estándares de Calidad del Software [2].

Nivel de calidad	Modelo de calidad	Estándar de calidad
Proceso	CMMi	ISO 9000:3
	TickIT	ISO 1220:7
	Bootstrap	ISO 15504 (SPICE)
	Personal SW Process (PSP)	IEEE /EIA 12207
	Team SW Process (TSP)	ISO 20000
	Practical SW Measurement (PSM)	ITIL Cobit 4.0
	Six Sigma for Software	
Producto	Gilb	ISO 9126-1
	GQM	ISO 25000 (SQUARE)
	Mc Call	IEEE Std 1061-1998
	Furps	
	Boehm	
	SATC	
	Dromey	
	C-QM	
	Metodología SQAE	
	WebEQM	

Luego de haber visto cada uno de los modelos con sus posibles métricas, se decidió enfocarse en el modelo CMMi puesto que actualmente en la Universidad los proyectos realizados en los diferentes centros se rigen por dicho modelo, el cual cabe recalcar se encuentra certificado en su nivel 2.

1.4 Pruebas de software

Las pruebas de software (en inglés *software testing*) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o *stakeholder*. Es una actividad más en el proceso de control de calidad. Son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno corresponde un nivel distinto de involucramiento en las actividades de desarrollo [5].

En este epígrafe se hace un análisis acerca de todo el proceso de pruebas de un software en desarrollo. Se abarcarán importantes aspectos como los principios por los cuales deben regirse la buena realización de pruebas; además de los niveles y tipos de pruebas que se deben conocer. Además, se presentarán los deberes y responsabilidades de cada uno de los involucrados en un equipo de desarrollo durante esta etapa.

1.4.1 Principios de las pruebas

Las pruebas se rigen por una serie de principios, una buena comprensión de estos facilitará el posterior uso de los métodos en un efectivo diseño de casos de prueba [6]. A continuación, se citan:

- **Principio 1- Las pruebas demuestran la presencia de defectos:** las pruebas pueden demostrar que hay defectos, pero no pueden demostrar que no los hay. Las pruebas reducen la probabilidad de que existan defectos ocultos en el software, pero, aunque no se detecte ningún defecto, no constituye una evidencia de corrección.
- **Principio 2- Las pruebas exhaustivas no existen:** probar todo (todas las combinaciones de entradas y precondiciones) es imposible, salvo en casos triviales. En lugar de pretender hacer pruebas exhaustivas, se deben realizar análisis de riesgos y prioridades para centralizar los esfuerzos de las pruebas.
- **Principio 3- Pruebas tempranas:** para identificar los defectos en una etapa temprana, las actividades de pruebas se iniciarán lo antes posible en el ciclo de vida del software o del desarrollo del sistema, debiendo centrarse en objetivos definidos.
- **Principio 4- Agrupación de defectos:** las pruebas deben concentrarse de manera proporcional en la densidad esperada y más tarde observada, de los defectos de los módulos. Normalmente la mayor parte de los defectos detectados durante las pruebas previas al lanzamiento y la mayoría de los fallos operativos, se concentran en un número reducido de módulos.
- **Principio 5- Paradoja del pesticida:** si repetimos las mismas pruebas una y otra vez, eventualmente a la misma cantidad de casos de prueba dejará de encontrar nuevos defectos. Para superar esto, los casos de prueba deben revisarse periódicamente y deben escribirse pruebas nuevas y diferentes para ejercitar distintas partes del software con vista a detectar más defectos.
- **Principio 6- Las pruebas dependen del contexto:** las pruebas se llevan a cabo de manera diferente según el contexto. Así, por ejemplo, la forma de probar un software crítico para la seguridad variará de la de un sitio de comercio electrónico.

- **Principio 7- Falacia de ausencia de errores:** La detección y corrección de defectos no servirá de nada si el sistema construido no es usable y no cumple las expectativas de los usuarios finales.

Estas leyes que definen básicamente la aplicación de las pruebas de software, en su totalidad tienen por objetivo ayudar a refinar el producto de software a través de las etapas involucradas.

1.4.2 Niveles de Prueba

Cuando se le van a aplicar pruebas a un software, se tienen en cuenta una serie de objetivos en diferentes escenarios y niveles de trabajo, debido a que las pruebas son agrupadas por niveles que se encuentran en distintas etapas del proceso de desarrollo.

A continuación, se muestran cada uno de los niveles de pruebas a tener en cuenta [6]:
Pruebas de componentes (también conocidas como pruebas de unidad, módulo o programa): tienen por objetivo localizar defectos en el software y comprobar el funcionamiento de los módulos del software, programas, objetos, clases, entre otros, que pueden probarse por separado. Pueden realizarse de manera independiente del resto del sistema, en función del contexto del ciclo de vida de desarrollo y del sistema. Dichas pruebas pueden incluir pruebas de funcionalidad y características no funcionales específicas, tales como, el comportamiento de recursos (por ejemplo, la búsqueda de filtraciones de memoria) o pruebas de robustez, además de pruebas estructurales (por ejemplo, cobertura de decisión). Por lo general son llevadas a cabo mediante el acceso al código objeto de las pruebas y con el soporte de un entorno de desarrollo; y por consiguiente los errores detectados se corrigen en el momento.

Pruebas de integración: se ocupa de probar las interfaces entre los componentes, las interacciones con distintas partes de un mismo sistema, como el sistema operativo, el sistema de archivo y el hardware, así como las interfaces entre varios sistemas. Puede haber más de un nivel dentro de estas pruebas, los cuales pueden llevarse a cabo en objetos de pruebas de distinto tamaño, según se indica a continuación:

- Las pruebas de integración de componentes se ocupan de probar las interacciones entre los componentes del software y se realizan a continuación de las pruebas de componente.
- Las pruebas de integración de sistemas se ocupan de probar las interacciones entre los distintos sistemas o entre el hardware y el software, y pueden realizarse a continuación de las pruebas de sistema.

Cuanto más amplio sea el alcance de la integración, más difícil será aislar los fallos de un componente o sistema específico, lo que puede provocar un mayor riesgo y un tiempo adicional de diagnóstico.

Pruebas de sistema: se refieren al comportamiento de todo un sistema/producto. El entorno de prueba debe coincidir en la máxima medida posible con el objetivo final o con el último entorno de producción a fin de minimizar el riesgo de no identificar fallos específicos del entorno durante las pruebas. Las pruebas de sistema pueden incluir pruebas basadas en riesgos y/o en especificaciones de requisitos, procesos de negocio, casos de uso u otras descripciones de texto de alto nivel o modelos del comportamiento del sistema, interacciones con el sistema operativo y recursos del sistema.

Pruebas de aceptación: su objetivo es crear confianza en el sistema, partes o características específicas no funcionales del sistema; o sea, su objetivo principal no es localizar defectos. Las pruebas de aceptación evalúan la buena disposición de un sistema para su despliegue y uso. En general estas pruebas pueden adoptar diferentes formas:

- Pruebas de aceptación de usuario: verifican la idoneidad de uso del sistema por parte de los usuarios comerciales.
- Pruebas operativas (de aceptación): la aceptación del sistema por parte de los administradores del sistema entre las que se incluyen: pruebas de backup/restauración, recuperación de desastres, gestión de usuarios, tarea de mantenimiento, carga de datos y tareas de migración, comprobaciones periódicas de vulnerabilidades de seguridad.
- Pruebas aceptación contractual y normativa: las pruebas de aceptación contractual toman como base los criterios de aceptación previstos en un contrato para fabricar un software desarrollado a la medida, dichos criterios deberán establecerse en el momento en que las partes involucradas aceptan el contrato; las pruebas de aceptación normativas toman como base cualquier normativa de obligado cumplimiento, tales como, normativas gubernamentales, legales o de seguridad.
- Pruebas alfa y beta (o de campo): las pruebas alfa se llevan a cabo en el emplazamiento de la organización de desarrollo, pero no las realiza el equipo de desarrollo; las pruebas beta (o pruebas de campo) las realizan los clientes o clientes potenciales en sus propias instalaciones.

1.4.3 Tipos de prueba

Las pruebas de software se dividen en dos grupos: las Pruebas Estáticas (sin uso de computadoras), son básicamente manuales, utilizadas para la revisión del código y planeamiento de los objetivos y fases de las pruebas; y las Pruebas Dinámicas (con uso de computadoras) utilizadas para probar aspectos tales como la funcionalidad, integración, resistencia al uso concurrente, entre otros. A continuación, se muestra detalladamente en que consiste cada una de las técnicas que pueden ser aplicadas asociadas a cada tipo de prueba.

Pruebas estáticas

Estas pruebas se basan en la revisión sistemática de artefactos tales como, el código y los documentos con las especificaciones y el diseño. Son técnicas realizadas en general por humanos, aunque es posible también utilizar autómatas para realizar ciertos análisis estáticos del código.
Las pruebas estáticas son muy efectivas para la prevención de defectos. Ha sido señalado en reiteradas fuentes en la literatura e Internet que estas técnicas se constituyen en la principal fuente de detección de errores [7].

Las inspecciones y recorridas son las dos técnicas más importantes de tipo estático realizables por personas. Se puede inspeccionar el código, los documentos de requerimientos u otros, procediendo de forma similar más allá del objetivo a verificar. El objetivo es comprobar la correspondencia entre cierta porción de código y su especificación funcional. A continuación, se describirán brevemente algunas de estas técnicas.

Inspecciones: Las inspecciones son exámenes visuales de un producto de software para detectar anomalías. Involucran necesariamente al autor del producto a revisar e incluyen un líder (facilitador o moderador) entrenado en técnicas de inspección. No deben participar los jefes de los participantes. Durante una sesión de inspección ocurren dos actividades principales [1]:

- El autor expone, leyéndole a los demás el contenido de un documento (leyendo línea por línea si se trata del código de un programa) mientras los otros intervienen con preguntas y diversas apreciaciones. El simple hecho de leer a una audiencia tiene una remarcable efectividad en la detección de errores. El moderador es responsable de asegurar que la discusión procede por vía productiva y enfocada en la detección de errores, no en su corrección.
- Se controla contra una lista (histórica) de errores comunes.

El tiempo óptimo para estas sesiones es de 90 a 120 minutos. Para que sean efectivas se debe establecer una actitud apropiada, donde el autor no debe sentir la inspección como un ataque a su trabajo.

Cuando la inspección la realiza el propio desarrollador sobre su trabajo se conoce como “auto inspección” o “revisión de escritorio”. No es tan efectiva como las inspecciones grupales [1].

Recorridas (Walk-throughs): Los objetivos de esta técnica se fijan en encontrar anomalías en el producto, mejorarlo, considerar soluciones alternativas y evaluar la conformidad con estándares y normativas. Es similar a una inspección, aunque conducida más informalmente. La organiza el líder del equipo para dar la oportunidad de revisar en forma conjunta el trabajo. La recorrida es dirigida por un

diseñador o programador y los participantes (miembros del equipo de desarrollo y otros interesados) intervienen haciendo preguntas, formulando críticas y esgrimiendo comentarios sobre probables defectos, errores, violaciones de estándares y otros [1].

Revisiones entre pares: Por "revisiones entre pares" se entiende el proceso de someter el trabajo de una persona al escrutinio de otro experto en la misma área, para que le revise y formule las apreciaciones que entienda conveniente. Aplicado al software puede lograrse, por ejemplo, intercambiando entre dos programadores su código para una revisión; o, en una escala de tiempo más inmediata y dinámica, trabajando en parejas [1].

Pruebas Dinámicas

Todas aquellas pruebas que para su ejecución requieren la ejecución de la aplicación. Las pruebas dinámicas permiten el uso de técnicas de caja negra y caja blanca con mayor amplitud. Debido a la naturaleza dinámica de la ejecución de pruebas es posible medir con mayor precisión el comportamiento de la aplicación desarrollada.

- Técnicas de caja blanca o estructurales; las cuales se basan en un minucioso examen de los detalles procedimentales del código a evaluar (ver Figura 2).
- Técnicas de caja negra o funcionales; se enfocan en realizar pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa (ver Figura 2).

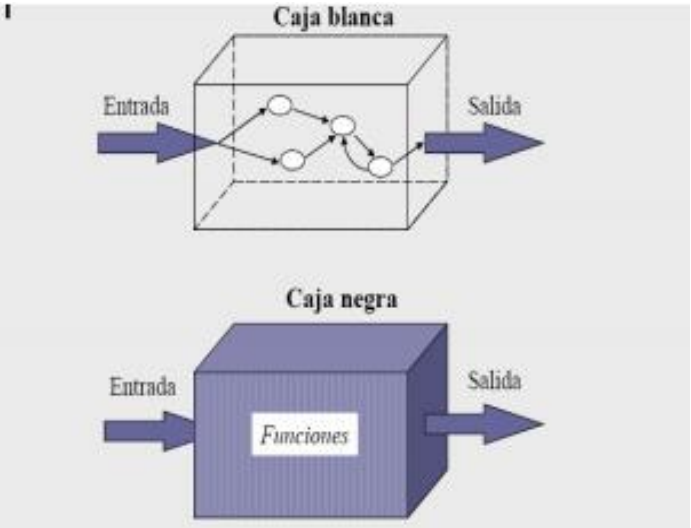


Figura 2: Representación de pruebas de Caja Blanca y Caja Negra [1].

A continuación, se listan los procedimientos propuestos por ambos tipos de técnicas para generar casos de prueba [1].

Técnicas de caja negra:

- Partición de equivalencia.
- Análisis de valores límites.
- Grafos de causa y efecto.

Técnicas de caja blanca:

- Camino básico.
- Flujo de datos.
- Bucles.

Luego de un exhaustivo estudio realizado de los diferentes tipos de pruebas, la autora de la presente investigación se enfoca en las pruebas estáticas, ya que estas permiten:

- Asegurar el progreso del proyecto, un correcto uso de los recursos y recomiendan acciones correctivas.
- Evalúan la conformidad de un producto con respecto a especificaciones y aseguran la integridad técnica y conceptual de los cambios.
- Proveen datos del producto y del proceso de revisión.
- Verifican que el producto cumple con los criterios predefinidos.

1.4.4 Roles en la disciplina de Pruebas

Para el proceso de Prueba se definen varios roles, los cuales cumplen una función específica dentro del equipo de proyecto y poseen varias habilidades en aras de desempeñar correctamente su trabajo. Es importante recalcar que para la línea de desarrollo de videojuegos estos roles son diferentes. Para ello el autor de la presente investigación considera que los roles involucrados serán: usuarios externos, usuarios finales, equipo de desarrollo. A continuación, se exponen sus responsabilidades y habilidades en la Tabla 2: Responsabilidades y habilidades de los roles dentro de la disciplina de pruebas [8].

Tabla 2: Responsabilidades y habilidades de los roles dentro de la disciplina de Pruebas [8].

Roles		
Analista	**Responsabilidades**	**Habilidades**
	• Participa con el cliente y el usuario final recogiendo las entradas de los involucrados relevantes.	• Habilidades de comunicación. • Capacidad de redacción y concreción.

	• Diseña las pruebas. • Participa en la elaboración del Plan de Administración de Requisitos (REQM). • Determina los proveedores válidos de requisitos (REQM). • Crea y actualiza las Matrices de Trazabilidad (REQM). • Elabora los Manuales de usuario. • Elabora Glosario de Términos.	• Habilidades de trabajo en equipo. • Conocimiento de metodologías de desarrollo de software. • Capacidad de trabajar con herramientas CASE. • Conocimiento de notaciones: UML BPMN, entre otras. • Conocimiento de patrones de requisitos, arquitectura y diseño. • Capacidad de aplicar Técnicas y métodos de prueba. Capacidad de aplicar métricas para determinar el tamaño y complejidad de los requisitos.
Diseñador de pruebas	**Responsabilidades**	**Habilidades**
	• Participa en la confección de la estrategia y el plan de pruebas. • Identifica los métodos, las técnicas, herramientas y directrices apropiadas para implementar las pruebas necesarias. • Establece en ambiente de comprobación. • Encargado de diseñar casos de pruebas para el sistema. • Dirige la definición del enfoque de prueba y garantiza la implementación satisfactoria. • Diseña las pruebas. • Genera los casos y datos de prueba. • Puede fungir como probador. • Analiza y evalúa el estado del producto de trabajo comprobado. • Evalúa el desempeño del probador y la calidad de las pruebas realizadas.	• Habilidad en el diseño de planes y casos de prueba. • Programación del liderazgo del equipo y habilidades de diseño de software (altamente deseable). • Experiencia en estimar esfuerzos de prueba (deseable). • Buenas habilidades analíticas; mente desafiante y curiosa; atención al detalle y tenacidad. • Comprensión de las anomalías de software y errores comunes. • Conocimiento del dominio (muy deseable). • Conocimiento del sistema o aplicación que se somete a prueba (muy deseable). • Habilidades de diagnóstico y resolución de problemas.

		• Amplio conocimiento de instalación y configuración de hardware y software. • Experiencia y éxito con la utilización de las herramientas de automatización de prueba. Habilidades de programación (preferible). Y todas las de un probador.
Probador	**Responsabilidades**	**Habilidades**
	• Encargado de seguir los planes de pruebas. • Ejecuta los casos de prueba y genera no conformidades asociadas al mismo. • Registra los resultados de las pruebas. • Analiza los resultados de las pruebas realizadas.	• 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 (deseable). • Conocimiento de la arquitectura de red y del sistema (deseable). Formación en la utilización apropiada de las herramientas de automatización de prueba. • Experiencia en la utilización de herramientas de automatización de prueba. Habilidades de programación. Habilidades de depuración y diagnóstico.
Administrador de calidad	**Responsabilidades**	**Habilidades**
	• Elabora el Plan de Aseguramiento de la Calidad. • Participa en la elaboración del Plan de Monitoreo y en el monitoreo y análisis de las áreas de procesos. • Elabora los planes de prueba.	• Trabajar en grupo. • Ser buen comunicador. • Ser líder. • Dominar técnicas estadísticas. • Poder de análisis estadístico.

	• Participa en las revisiones técnicas formales de los productos de trabajo. • Participa en las revisiones con el cliente de los entregables. • Guía el diseño y ejecución de las pruebas internas. • Participa en el análisis y recolección de los datos para las mediciones. • Vela por el cumplimiento de las políticas de la organización y reglas bases del proyecto. • Colabora en las auditorías que se realicen al proyecto. • Coordina y colabora con las pruebas de liberación externa al proyecto. • Crea una cultura de calidad en el proyecto. Realiza las revisiones de inconsistencias y monitorea las no conformidades hasta su cierre (REQM).	• Dominar técnicas de recolección de información. • Poder de síntesis. • Ser persuasivo. • Dominar el ciclo de desarrollo de software. • Dominar las materias de ingeniería y gestión de software. • Dominar la programación. • Dominar el modelo CMMI. • Explotar con efectividad la suite de office. • Dominar los tipos de pruebas. • Dominar explotar herramientas de automatización de pruebas. • Conocer los principales estándares internacionales en la producción de software, así como los procedimientos y lineamientos que norman la producción en la UCI. • Técnica de dirección. • Toma de decisiones. • Conocer los principales conceptos relacionados con la calidad de software. • Dominar los principios de gestión de la calidad.
Equipo de desarrollo	**Responsabilidades**	**Habilidades**
	• Actualizar el estado de un elemento de configuración. • Aceptar la asignación de la Solicitud de cambio. • Implementar el cambio. • Corregir errores encontrados en las líneas bases. • Resolver las No Conformidades asignadas. • Participar en la recolección de los datos para las mediciones.	• Es recomendable que las personas que ocupen este rol sean graduados de alguna de las especialidades relacionadas con la informática o las ciencias de la computación. En su defecto puede desempeñarse alguna persona con experiencia en la actividad de desarrollo de sistemas

	• Revisar los diferentes indicadores monitoreados.	informáticos.
Usuarios internos	**Responsabilidades**	**Habilidades**
	• Opera el sistema (videojuego). • Informa acerca de inconformidades en el sistema. • Aporta sugerencias acerca del mejoramiento de algún procedimiento en el sistema.	• Conocimiento del dominio de la aplicación.
Usuarios externos	**Responsabilidades**	**Habilidades**
	• Revisa y aprueba los entregables del proyecto. • Firma los documentos legales del proyecto (Acta de Inicio, Especificación de Requisitos, Acta de Aceptación, Acta de terminación del proyecto).	• Habilidades de comunicación. • Capacidad de decisión, conocimientos del negocio.

1.5 Pruebas en proyectos de software

El proceso de Pruebas en proyectos de software se realiza durante cuatro etapas: Pruebas internas (PI), Pruebas de Liberación (PL) siguiendo lo establecido por el Sistema de Gestión de Calidad UCI, Pruebas de aceptación (PA) y Pruebas piloto (PP).

Inicialmente luego de la disciplina de implementación se generan los Diseños de Casos de Prueba (DCP) y en muchas ocasiones pueden ser generados en la disciplina de análisis y diseño. Una vez diseñados los Casos de Prueba (CP), se procede a planificar el período de las PI acordado por los miembros del proyecto. Dentro de etapa se ejecutan varias iteraciones, en una primera iteración se analiza la correspondencia existente entre los DCP y lo que está implementado. Además, se analiza que estén correctamente descritas cada una de las entradas de los CP para ello debe haber coherencia entre cada uno de sus elementos, no deben de existir faltas de ortografía, entre otros aspectos. Los defectos registrados asociados tanto a la aplicación como al DCP deben de ser registrados en el Registro de defectos y dificultades (RDD) con el objetivo de que el coordinador de las pruebas realice una conciliación de todas las no conformidades, no sin antes, haber verificado que las no conformidades se repitan. Acto seguido de esto el coordinador de pruebas registra las no conformidades en el módulo de alcance y calidad de la Suite de Gestión de Proyectos (GESPRO), donde se les asignan tareas, un tiempo de resolución y un encargado el cual se ocupará de dar solución a estas no conformidades. Una vez registradas, se les brinda un seguimiento sistemático mediante las reuniones de chequeo de proyectos o reuniones extras para el análisis del estado del proyecto. A la par, se le van realizando pruebas asociadas a la documentación, y por lo general los productos de trabajo que son revisados son los DCP y los manuales de usuario o configuración los cuales deben concebirse bajo una estrategia marcaria definida de acuerdo

a la línea temática del proyecto. También cabe recalcar que el encargado asignado a cada una de las no conformidades registradas cumpla con el tiempo establecido para su resolución. Luego de finalizadas las PI, el proyecto debe generar un acta de constancia que evidencien las pruebas realizadas, el equipo de trabajo que ejecutaron dichas pruebas y los productos de trabajo que pueden pasar a una evaluación más rigurosa a nivel de universidad.

De una manera particular los videojuegos que son desarrollados en el centro Vertex ejecutan las pruebas a diferentes niveles (alfa y beta), donde se chequean el correcto funcionamiento de todos los mecanismos definidos para el videojuego de forma íntegra; almacenándose de igual forma en el RDD. El procedimiento que resta se desarrolla de igual forma que en las otras líneas de desarrollo.

Después de haber realizado las PI se debe asistir a una evaluación del producto a un nivel un poco más minucioso, la cual es realizada en un Laboratorio de Pruebas de Software (LPS) donde se tiene una reunión inicial con los implicados del proyecto en esta actividad. Deben de ser entregados los documentos de Especificación de requisitos, DCP y un acta de constancia de la realización de las PI. En una primera iteración se comienzan a probar las funcionalidades del sistema contra los DCP realizados, luego de concluida esta iteración se realiza el mismo proceder para la determinación de las no conformidades correspondientes a ese momento. Estas no conformidades son entregadas al líder del proyecto y se acuerda un tiempo de entrega de los productos de trabajo en los cuales fueron identificadas las no conformidades, ya sean de la aplicación o de DCP. En una segunda iteración se realiza una regresión para analizar el estado de las no conformidades; conjuntamente se inspecciona la documentación asociada a la ayuda e instalación del sistema. En caso de detectarse otras no conformidades, se debe proceder a una tercera iteración y luego de esta a una prueba final. Siendo de esta manera, si el producto queda libre de defectos entonces es emitida por la dirección del LPS y el coordinador de las pruebas un acta de liberación.

Con el producto ya liberado, se procede con las PA en donde el producto es entregado al cliente en aras de ser comprobado su correcto funcionamiento. Seguidamente de esto el cliente debe de firmar un acta en donde expresa que está conforme con el producto.

1.5.1 Pruebas en videojuegos

De los distintos niveles de pruebas abordados en el acápite 1.4.2 para los proyectos que se encuentran sobre la línea de desarrollo de videojuegos, las pruebas que se le realizan son las pruebas de aceptación.

Para concretar las pruebas alfa y beta contenidas dentro de este nivel, con el objetivo de que los usuarios finales o externos prueben el producto e informen inconformidades detectadas.

1.6 Productos de trabajo de la disciplina de Pruebas

En el siguiente epígrafe se hace referencia a los diferentes productos de trabajo que deben de ser generados durante la disciplina de Pruebas y que se encuentran contenidos dentro del Expediente de proyectos de desarrollo UCI; el cual cabe recalcar que en estos momentos se encuentra en su versión 4.0.

1.6.1 Productos de trabajo según el Expediente de proyecto de desarrollo 4.0 UCI

El análisis de dicho expediente es importante para el desarrollo de la propuesta de solución, ya que este contiene los productos de trabajo generados durante el proceso de Ingeniería de Requisitos (IR^2) en la universidad. Este expediente cuenta con 4 acápites fundamentales en los que se encuentran distribuidos los diferentes artefactos, estos son: Ingeniería, gestión de proyecto, soporte y líneas bases. Teniendo en cuenta el enfoque hacia el cual se inclina la propuesta de solución se presta total atención sobre los productos de trabajo concernientes a la disciplina de Pruebas. Estos productos de trabajo son los que en su momento serán generados de forma semiautomática e interactiva por el sistema en desarrollo.

Los productos de trabajos estudiados son:

- **Lista de verificación de revisiones de inconsistencias**: Este artefacto es una lista de chequeo cuyo objetivo principal es detectar las inconsistencias entre los elementos de trazabilidad de los requisitos del software.
- **Diseño de Casos de Prueba**: Este artefacto es la especificación de un conjunto de variables de entradas, condiciones de ejecución y resultados esperados, los cuales son identificados con el propósito de realizar una evaluación de un aspecto particular en un escenario específico.
- **Registro de defectos y dificultades detectados**: Este artefacto recolecta los resultados maliciosos encontrados durante la ejecución de una o más pruebas durante un ciclo completo de pruebas.

A continuación, se muestra una tabla descriptiva con los principales elementos que conforman a cada uno de los productos de trabajo expuestos con anterioridad:

Tabla 3: Descripción de los productos de trabajo.

Producto de trabajo	Descripción
Diseño de Casos de Prueba	• Por cada Caso de Uso (CU) se debe de conformar un Caso de Prueba (CP). • Las secciones incluidas dentro de un CP estarán en correspondencia con cada una de las secciones que se hayan registrado en el CU correspondiente. • Está compuesto por escenarios que constituyen el flujo básico y los diferentes flujos alternos del CU en cuestión. • Por cada sección se deben de registrar las variables que intervienen en el escenario. • Para los escenarios se realiza una breve descripción y para las variables se le debe de asignar un valor arbitrario que cumpla con las condiciones del escenario, con el objetivo de verificar que el sistema opera correctamente. • Posee un campo denominado “Respuesta del sistema”, en el cual deberá quedar plasmada la acción que emite el sistema al comprobar los distintos escenarios. • También posee un campo denominado “Flujo central”, en el cual se reflejan los pasos que se deben seguir para la correcta verificación de la funcionalidad que corresponde al escenario. • Por otra parte, también se debe de tener una descripción de las variables presentes en el CP; para ello se debe de tener en cuenta su clasificación, la cual se dará en correspondencia con el componente de diseño utilizado. También se debe de especificar si el campo puede ser o no nulo y por último una breve descripción de los datos que deben de ser introducidos en dicho campo.
Lista de verificación de revisiones de inconsistencias	• Posee una serie de evidencias ya definidas; aunque si se considera que el producto de trabajo debiera de contener otras evidencias además de las ya especificadas pueden ser agregadas. • Con el objetivo de evaluar dichas evidencias se definen varios campos tales como: *el impacto: puede ser evaluado de alto, medio o bajo (A, M, B).

	*la ubicación: se especifica la ubicación de la evidencia a evaluar. *uso: especifica si es de uso obligatorio o condicional. *procedimiento: define cómo cumplir con la evidencia a evaluar.
Registro de defectos y dificultades detectados	• Recoge todas las inconsistencias y defectos arrojados por los CP. • Presenta varios campos con el objetivo que queden evaluadas cada una de las inconsistencias detectadas durante la fase de prueba, tales como: *impacto: puede ser evaluado de alto, medio o bajo (A, M, B). *tipo de error: se clasifica en base a una lista de errores ya predefinidas. *descripción: se expone una breve descripción de la inconsistencia detectada. *respuesta del sistema: se especifica la respuesta que emite el sistema en cuanto a la inconsistencia detectada. *relación: evidencia la relación existente entre las inconsistencias detectadas y los elementos de trazabilidad del software.

1.7 Herramientas para la gestión de pruebas

Selenium. Compuesto por dos herramientas: Selenium IDE y SeleniumWebDriver. La primera, un plugin de Firefox que te genera un entorno de desarrollo y que permite crear casos de prueba para aplicaciones web. La segunda, Selenium WebDriver, ejecuta las pruebas. Este entorno de pruebas automáticas opera en los principales navegadores (InternetExplorer, Mozilla, Chrome y Opera). Además, permite pruebas para dispositivos móviles, para iPhone y Android. Utiliza los siguientes lenguajes: Python, Ruby, Java y C#. La licencia es “Apache 2.0 License”.

- Generación de CP.
- Exportar CP en formato de lenguaje de programación Selenium RC [9].

JMeter. Aplicación de escritorio en Java y dentro del proyecto Jakarta. Esta herramienta permite realizar pruebas funcionales (y de rendimiento) para aplicaciones web. Trabaja con los siguientes protocolos: HTTP, HTTPS, SOAP, JDBC, LDAP, JMS, Mail – POP3(S) and IMAP(S). La licencia es “Apache 2.0 License”.

- Diseño y ejecución de testplan.
- Genera ficheros. jmx

Testlink. Permite crear y gestionar casos de prueba, organizarlos en planes de pruebas, realizar un seguimiento de los resultados, establecer trazabilidad con los requisitos, generar informes, entre otros. Se integra con otros sistemas de seguimiento de "*bugs*" y "*ticketing*" como Bugzilla, Mantis, entre otros. Licencia: GPL. Es una aplicación web hecha en php por lo cual se requiere tener una máquina con php+mysql como server y los usuarios accederán a Testlink a través de un browser. Además, nos ofrece:

- Soporte para diferentes productos.
- Creación de diferentes roles con distintos permisos para los integrantes del equipo de testing.
- Creación ilimitada de carpetas en forma de árbol (llamadas testsuites) para una mejor organización y agrupamiento de los casos de prueba.
- Versionado de casos de prueba.
- Ejecución de los casos de prueba por versión del software bajo prueba.
- Creación de testplans para la ejecución.
- Asignación de casos (para ejecución) a usuarios. De modo que en un mismo test plan podemos tener varios *testers* trabajando y ellos sólo ven sus casos de prueba asignados.
- Soporte para linkear casos con requerimientos/especificaciones.
- Generación de distintos tipos de reportes: generales del producto, por testplan, gráficos, entre otros.
- Posibilidad de mandar los reportes por mail al *tester* [10].
- Exporta informes en XML y CSV.
- Asigna requisitos a los CP.
- Posee una sección específica para gestionar requisitos, donde se puede asignar los requisitos a los CP.
- Se puede importar/exportar CP.

Luego de realizado un análisis de las herramientas expuestas con anterioridad, la autora establece que las mismas pueden servir como base para la realización de la solución a desarrollar. Aunque, en dichas herramientas no existen algunas funcionalidades críticas que son vitales a tener en cuenta para la propuesta de solución durante la disciplina de Prueba, tales como:

- La generación de los productos de trabajo de la disciplina de Pruebas "Lista de verificación de revisiones de inconsistencias" y "Registro de defectos y dificultades detectados".
- La generación de Casos de Pruebas a partir de Casos de Uso especificados en previas disciplinas.
- Exportar los diferentes productos de trabajo en formato Excel.

Teniendo esto en cuenta, se plantea que dichas herramientas no poseen en su totalidad la capacidad para dar solución al problema de investigación expuesto inicialmente. Esto se debe a que constituyen herramientas enfocadas a la realización de pruebas dinámicas; mientras que el objetivo general del presente trabajo está enfocado hacia las pruebas estáticas, las cuales se centran en parte en el análisis documental.

1.8 Tendencias y tecnologías

En el siguiente epígrafe se hace un análisis de las diferentes tecnologías, herramientas y metodología a utilizar en el desarrollo de la aplicación resultante de esta investigación; con el fin de seleccionar las más apropiadas para el desarrollo de la aplicación web, teniendo en cuenta sus prestaciones. A continuación, se especifican detalles sobre estos elementos:

1.8.1 Metodología de desarrollo

Existen metodologías ágiles y robustas. Las metodologías ágiles son apropiadas para guiar proyectos de poco volumen que requieran una rápida implementación, un ejemplo es Extremme Programming (XP). Las metodologías robustas pueden ser empleadas para guiar el proceso de desarrollo de proyectos grandes o pequeños, aunque son más apropiadas para proyectos grandes que por su importancia requieren una fuerte planificación y generación de documentación, ejemplo RUP (Rational Unified Process). Este proyecto tiene gran importancia, pues, se enfoca en la gestión semi-automatizada de los productos de trabajo que se generan durante el proceso de prueba, por lo que se requiere de una metodología capaz de guiar el proceso con precisión, que contribuya a obtener un software fiable, sin errores [11].

Proceso Unificado Ágil (AUP, por sus siglas en inglés) es la metodología que se ajusta a la necesidad del proyecto porque combina características de la metodología ágil XP con los artefactos de RUP [12]. Dentro de las características particulares de AUP, se tiene que es una versión simplificada de la metodología RUP. La Figura 3 muestra el ciclo de vida de esta metodología. AUP abarca siete flujos de trabajos, cuatro ingenieriles y tres de apoyo: Modelado, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyectos y Ambiente respectivamente. El Modelado agrupa los tres primeros flujos de RUP (Modelamiento del negocio, Requerimientos y Análisis y Diseño). Dispone de cuatro fases igual que RUP: Inicio, Elaboración, Construcción y Transición.

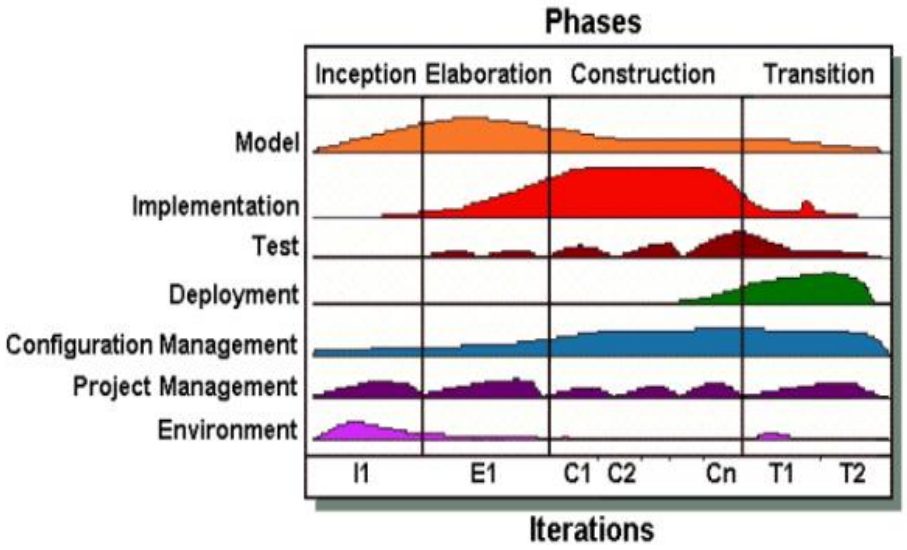


Figura 3: Esquema de los flujos de trabajo y fases de AUP [13].

El Modelado es el flujo de trabajo que tiene el objetivo de entender el negocio de la organización, el problema de dominio que se aborda en el proyecto y determinar una solución viable para resolver el problema de dominio. El flujo de trabajo Implementación tiene como objetivo transformar sus modelos en código ejecutable y realizar un nivel básico de las pruebas, en particular, la unidad de pruebas. El flujo de trabajo de Prueba tiene como objetivo realizar una evaluación objetiva para garantizar la calidad. Esto incluye la búsqueda de defectos, validar que el sistema funciona tal como está establecido, verificando que se cumplan los requerimientos [12]. Por último, dentro de los flujos de trabajo ingenieriles se tiene el Despliegue, cuyo objetivo es el plan para la prestación del sistema y la ejecución de dicho plan, para que el sistema quede a disposición de los usuarios finales. Esta versión ágil de la metodología RUP se basa en los siguientes principios:

- Simplicidad: refiere a que todo se describe de forma concisa usando poca documentación, no muchas de ellas.
- Agilidad: refiere al ajuste de los valores y principios de la Alianza Ágil. Es por ello que es necesario centrarse en actividades de alto valor: La atención se centra en las actividades que en realidad lo requieren, no en todo el proyecto.
- Herramienta de la independencia: refiere a que se puede usar cualquier conjunto de herramientas que desea con el AUP. Sugiere utilizar las herramientas idóneas para el trabajo, que normalmente son herramientas simples o incluso herramientas de código abierto.
- Usted querrá adaptar este producto para satisfacer sus propias necesidades: refiere a que la metodología AUP es un producto de fácil uso, indiferentemente de la herramienta usada [12].

1.8.2 Lenguaje de modelado

El Lenguaje Unificado de Modelado (en lo adelante UML, por sus siglas en inglés, *Unified Modeling Language*) se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de *software*. Se emplea para diseñar, hojear, configurar, mantener y controlar la información sobre sistemas. Está pensado para emplearse con todos los métodos de desarrollo, etapas del ciclo de vida y dominios de aplicación. UML incluye conceptos semánticos, notación y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código, así como generadores de informes [14]. Es importante resaltar que UML es un lenguaje para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de *software*, para detallar los artefactos en el mismo, para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de *software*, pero no especifica en sí mismo qué metodología o proceso utilizar.

1.8.3 Herramienta de modelado

Las herramientas de ingeniería de *software* asistida por computadoras (en lo adelante CASE, por sus siglas en inglés, *Computer Aided Software Engineering*) brindan ayuda a los analistas, ingenieros de *software* y desarrolladores. Por ejemplo, permiten el modelado de los sistemas mediante diferentes diagramas, generación de código a partir de estos y viceversa. Las herramientas CASE de modelado con UML permiten aplicar la metodología de análisis y diseño orientados a objetos. Además, permiten abstraerse del código fuente, en un nivel donde la arquitectura y el diseño se tornan más fáciles de entender.

La herramienta recomendada a utilizar para el modelado de la aplicación es Visual Paradigm, debido fundamentalmente a que es una herramienta multiplataforma que ayuda a una rápida construcción de aplicaciones de calidad. Además, la UCI posee licencia para su utilización. A continuación, se brindan detalles de esta herramienta.

Visual Paradigm

Visual Paradigm para UML es una herramienta CASE profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Brinda

la posibilidad de generar código a partir de los diagramas para lenguajes como Java y PHP, así como obtener diagramas a partir de código.

1.8.4 Lenguaje de programación para aplicaciones web

Para la implementación de la aplicación se propone usar el lenguaje PHP. Esto se debe a que el sistema AREXPRO, al cual debe de estar integrado la propuesta de solución a desarrollar se encuentra implementado bajo este lenguaje.

PHP

"Es un lenguaje que es independiente de plataforma, con una gran librería de funciones que cubren desde cálculos matemáticos complejos hasta tratamiento de conexiones de red. Es un lenguaje conocido mundialmente que ofrece disimiles ventajas entre las que se destacan:

- Completamente orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una Base de Datos.
- El código fuente escrito en PHP es invisible al navegador y al cliente ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea segura y confiable.
- Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL" [15].

1.8.5 Framework de desarrollo y presentación

El desarrollo de aplicaciones *web* está actualmente condicionado por la utilización de los marcos de trabajo; si se concibe un producto medianamente grande es muy recomendable la utilización de un *framework*. Un *framework*, es una estructura de soporte definido, mediante la cual otro proyecto de software puede ser organizado y desarrollado. "Es una estructura conceptual y tecnológica compuesta por bibliotecas, componentes y clases que facilitan el desarrollo ágil, seguro y escalable de una aplicación" [16]. Simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Ofrecen una infraestructura que permite tener un código ordenado, limpio y fácil de actualizar, un código seguro, robusto y eficiente.

Como *framework* se propone utilizar el *framework* Yii. Debido principalmente a que el sistema AREXPRO, al cual debe de estar integrado la propuesta de solución a desarrollar se encuentra implementado bajo este marco de trabajo.

Yii: Es un framework PHP basado en componentes de alta performance para desarrollar aplicaciones Web de gran escala. El mismo permite la máxima reutilización en la programación web y puede acelerar el proceso de desarrollo. Posee una variedad de características, tales como:

- Posee una curva rápida de aprendizaje, tiene una excelente comunidad, foros y demás.
- Utiliza el patrón Modelo-Vista-Controlador (MVC) de una manera fácil de entender y muy organizado. Posee una excelente documentación.
- Cuenta con un módulo Gii para la autogeneración de código, no sólo del modelo, sino que tiene *scafolding* el cual te genera vistas y controladores (CRUD). Además, puedes generar o definir tus propias plantillas para esa autogeneración. En caso de no gustar las plantillas que trae por defecto, se les puede cambiar el estilo y la estructura de los directorios por lo que se convierte en un framework bastante flexible [17].

1.8.6 Gestor de base de datos

PostgreSQL: Es un sistema de gestión de bases de datos objeto-relacional. Está liberado bajo licencia BSD, además es extensible, multiplataforma y presenta modelos de negocios rentables con instalaciones a gran escala [18]. Tiene ahorros considerables en costos de operación. El *software* ha sido diseñado y creado para tener un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características de rendimiento.

Se utiliza PostgreSQL puesto que es el gestor de base de datos sobre el cual se encuentra implementado el sistema AREXPRO sobre el cual se integrará la propuesta de solución anteriormente. Además, permite usar procedimientos almacenados, tiene un rendimiento medio, es una tecnología libre, multiplataforma y puede ser utilizado con varios entornos de desarrollo. También es de código abierto y es el más usado en la universidad y en los productos que se desarrollan en el centro.

1.8.7 Entornos de desarrollos integrados

Un Entorno de Desarrollo Integrado (IDE, por sus siglas del inglés *Integrated Development Environment*) es un entorno de programación que ha sido empaquetado como un programa de aplicación. Consiste en un editor de código, un compilador, un depurador y en ocasiones un constructor de interfaz gráfica de

usuario. Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes [19]. Una decisión importante a la hora de desarrollar con PHP es la selección del IDE, ya que el entorno de desarrollo que se elija puede suponer una verdadera diferencia en el tiempo de trabajo invertido. Debido a lo anterior se realiza un estudio sobre algunos de los IDE para PHP:

- **Zend Studio:** es el soporte en desarrollos y pruebas de PHP con el set más completo de herramientas para la creación de aplicaciones altamente fiables como lo requiera una empresa. Asegura el desarrollo de *software* mediante la combinación del IDE con un entorno de prueba que agiliza la seguridad de la calidad, integración y las etapas de los procesos. Brinda todo lo que necesita para construir, probar y entregar aplicaciones de alto rendimiento. Contiene distintas funcionalidades como generación de código, formateo de código configurable, pruebas unitarias y desarrollo remoto. Cuesta 299€ y es multiplataforma (Linux, Windows y Mac OS X) [20].
- **NetBeans:** es multiplataforma (Windows, Linux, Mac OS X y Solaris), gratuito, de código abierto (con licencia *CDDL*) y se puede utilizar para programas en otros lenguajes además de PHP. A parte de las funciones básicas con las que debería contar cualquier IDE, como resaltado de sintaxis, autocompletado, formateo de código o depurador (xDebug), también cuenta con otras funcionalidades menos comunes como la integración con PHPUnit para las pruebas unitarias y con CVS, Subversion y Mercurial para el control de versiones [21].
- **PhpStorm:** JetBrains PhpStorm es un IDE multiplataforma comercial para PHP desarrollado sobre la plataforma JetBrains IntelliJ IDEA. Un IDE que tiene un número de funcionalidades similar al Netbeans, pero la licencia personal cuesta 88€ y la comercial 176€. Proporciona un editor para PHP, HTML y JavaScript con el análisis de código sobre la marcha, la prevención de errores y refactorizaciones automáticas para PHP y JavaScript. Ofrece soporte para PHP 5, incluyendo generadores, lista en foreach, espacios de nombres, los cierres, los rasgos y la sintaxis de matrices corto. Incluye un editor de pleno derecho de SQL con resultados de la consulta editables [22].

Atendiendo al estudio anterior, el IDE para la implementación recomendado es NetBeans debido a que está disponible para Windows, Mac, Linux y Solaris, es de código abierto, gratuito y sin restricciones de uso. Existe además un número importante de módulos para extenderlo y que permiten a los desarrolladores crear rápidamente aplicaciones *web* empresariales, de escritorio y aplicaciones móviles, utilizando PHP, así como Java, JavaScript, Ajax, Groovy, Grails, C / C ++, XHTML y soporte para CSS 3. Puede soportar *frameworks* y está respaldado por una fuerte comunidad de desarrolladores. Ofrece una variada selección de *plugins* de terceros, incluyendo uno para trabajar con el *framework* Yii [21].

1.8.8 Generador de reporte

Para exportar los productos de trabajo creados se utiliza la extensión del *framework* Yii PHPExcel.

PHPExcel: Es una librería que permite la escritura y lectura de formatos como Excel, Excel 2007, Office Open XML, entre otros. Es de código abierto, bajo la licencia LGPL, por lo que puede ser usado sin ningún tipo de restricción.

Posee un sin número de características, las cuales se listan a continuación [23]:

- Setear meta datos (autor, título, descripción, entre otros).
- Agregar más hojas de cálculo (pestañas).
- Agregar valores y fórmulas a celdas individualmente.
- Combina celdas.
- Proteger rango de las celdas con contraseña.
- Poder establecer ancho y alto de las celdas.
- Poder establecer diseño de celdas (fuente, bordes, fondo, entre otros).
- Soporte para fijar celdas.
- Poder agrupar celdas/columnas.
- Insertar o borrar celdas/columnas.
- Asignar nombre a un rango.
- Soporte para referencia entre pestañas.
- Agregar imágenes (con la posibilidad de cambiar su estilo).
- Agregar comentarios a una celda.
- Opciones de salida en diferentes formatos (Excel 2007, CSV, PDF, HTML, entre otros).
- Opciones de lectura en distintos formatos (Excel 2007, CSV, entre otros).

1.9 Conclusiones parciales

Después de haber realizado un estudio del estado del arte de la disciplina de Pruebas, algunas de las herramientas para la automatización de pruebas, los principales framework para PHP, los SGBD, los IDE, las herramientas CASE y las metodologías de desarrollo de software se concluye que:

- El análisis de los principales conceptos asociados al dominio del problema permitió tener una mejor comprensión de la solución informática a desarrollar. A partir de esto se resume que la disciplina de Pruebas tiene como principal objetivo descubrir defectos en el sistema antes que este sea entregado a su usuario final.
- El análisis de algunas de las herramientas para la automatización de pruebas, permitió corroborar que son herramientas que en su mayoría no cumplen en su totalidad con las necesidades de la solución propuesta.
- El estudio de las principales herramientas, tecnologías y metodologías permitió seleccionar las más adecuadas para el desarrollo del sistema.

Capítulo #2. Características y diseño del sistema

2.1 Introducción

En el presente capítulo se realiza la descripción de la propuesta de solución y se detallan los principales requerimientos de la aplicación. Además, se explica el uso de los patrones de diseño para la implementación y el patrón arquitectónico a aplicar. Por otra parte, se especifican cada uno de los casos de uso del sistema definidos según los requisitos identificados. Por último, se diseña el modelo de datos de la base de datos creada y el diagrama de clases de diseño.

2.2 Descripción de la propuesta de solución

La propuesta de solución consiste en un módulo para la automatización de los productos de trabajo del Expediente de proyecto de desarrollo en su versión 4.0. El mismo cuenta con la posibilidad de exportar en formato Excel cada uno de los productos de trabajo relacionados con la disciplina de Pruebas. Además, cuando el mismo se integre con el sistema AREXPRO, contará con la posibilidad de un sistema de autenticación basado en los diferentes roles (Administrador de Prueba, Administrador de Calidad, Probador), de esta manera se diferencia el entorno de trabajo correspondiente a cada rol.

El Administrador Analista (usuario que se desempeña bajo los roles de administrador de prueba y administrador de calidad) se encarga de la gestión de los CP, los registros de defectos y dificultades; y las listas de verificación de revisión de inconsistencias; dígase insertar, eliminar, modificar, mostrar y listar. Además, también se ocupa de generar los productos de trabajo pertenecientes a la disciplina de Pruebas (Diseño de Casos de Prueba, Registro de defectos y dificultades detectados, Lista de verificación de revisión de inconsistencias). También en la gestión del registro de defectos está involucrado el Probador, el cual como tal se encuentra generando a la vez el producto de trabajo asociado.

2.3 Modelo de dominio

Un modelo de dominio, es una representación de las clases conceptuales del mundo real, no de componentes de software. No se trata de un conjunto de diagramas que describen clases de software u objetos de software con responsabilidades. Es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. Podría considerarse como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio [24].

Todos los modelos en sí, son aproximaciones del dominio que estamos intentando entender. Un buen modelo del dominio captura las abstracciones e informaciones esenciales necesarias para entender el

dominio en el contexto de los requisitos actuales, y ayuda al personal pertinente a entender el dominio (sus conceptos, terminología y relaciones).

A continuación, se muestra el modelo de dominio de la propuesta de solución a desarrollar:

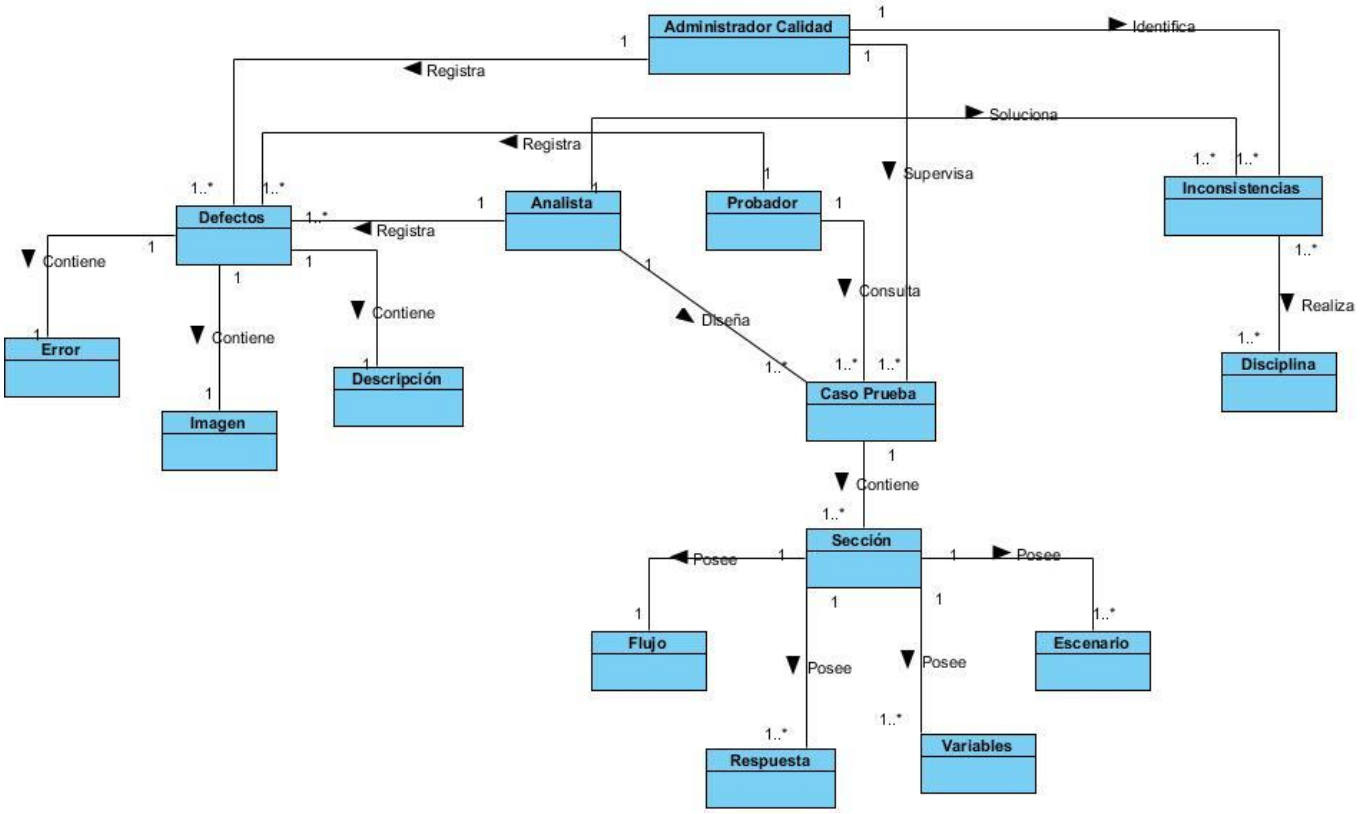


Figura 4: Modelo de dominio.

Tabla 4: Descripción de entidades.

Entidad	Descripción
Administrador Calidad	Es quien identifica inconsistencias, registra defectos detectados, y supervisa los casos de prueba.
Analista	Es quien soluciona las inconsistencias encontradas, diseña los casos de prueba y también se encarga de registrar los defectos que puedan ser detectados.
Probador	Es quien consulta los casos de prueba una vez diseñados, y también registra los defectos que se puedan ser detectados.
Defectos	Está compuesto por varios elementos fundamentales para que puedan ser registrados de forma comprensible, algunos de sus elementos son: Error, Imagen, Descripción.

Caso Prueba	Está compuesto por varias secciones, en las cuales se recogen elementos fundamentales tales como: Flujo, Respuesta, Variables, Escenario

2.4 Actores del sistema

El módulo informático resultante de este trabajo a los miembros del centro VERTEX en aras de contribuir en gran medida al proceso llevado a cabo durante la disciplina de Pruebas en un proyecto cualquiera. A continuación, se describen los actores de la aplicación web:

Tabla 5: Actores del sistema.

Actor	Descripción
Administrador Analista	Es el encargado (a) de gestionar los usuarios que interactuarán con el sistema, los CP, las listas de verificación de revisiones de inconsistencias y los registros de defectos y dificultades detectados. Además de se encargará de generas los productos de trabajo pertenecientes a la disciplina de Pruebas.
Probador	Es el encargado (a) de gestionar los registro de defectos y dificultades detectados y también de generar el producto de trabajo asociado al mismo.

2.5 Requisitos del sistema

“Los requerimientos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requerimientos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información.” [25].

2.5.1 Requisitos Funcionales

“Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer.” [25].

Tabla 6: Requisitos funcionales.

Nº	Nombre
RF1	Crear producto de trabajo “Registro de defectos y dificultades detectados”.
RF2	Modificar producto de trabajo “Registro de defectos y dificultades detectados”.
RF3	Consultar producto de trabajo “Registro de defectos y dificultades detectados”.

RF4	Exportar producto de trabajo “Registro de defectos y dificultades detectados”.
RF5	Crear iteración.
RF6	Eliminar iteración.
RF7	Consultar iteración.
RF8	Insertar defectos.
RF9	Modificar defectos.
RF10	Consultar defectos.
RF11	Eliminar defectos.
RF12	Crear producto de trabajo “Lista de verificación de revisión de inconsistencias”.
RF13	Modificar producto de trabajo “Lista de verificación de revisión de inconsistencias”.
RF14	Consultar producto de trabajo “Lista de verificación de revisión de inconsistencias”.
RF15	Crear lista de verificación.
RF16	Eliminar lista de verificación.
RF17	Consultar lista de verificación.
RF18	Listar casos de prueba.
RF19	Exportar producto de trabajo “Lista de verificación de revisión de inconsistencias”.
RF20	Crear evidencia de la lista de verificación de inconsistencias.
RF21	Modificar evidencia de la lista de verificación de inconsistencias.
RF22	Consultar evidencia de la lista de verificación de inconsistencias.
RF23	Eliminar evidencia de la lista de verificación de inconsistencias.
RF24	Evaluar evidencia de la lista de verificación de inconsistencias.
RF25	Crear producto de trabajo “Diseño de Caso de Prueba”.
RF26	Modificar producto de trabajo “Diseño de Caso de Prueba”.
RF27	Consultar producto de trabajo “Diseño de Caso de Prueba”.
RF28	Eliminar producto de trabajo “Diseño de Caso de Prueba”.
RF29	Exportar producto de trabajo “Diseño de Caso de Prueba”.
RF30	Buscar caso de prueba.
RF31	Generar listado de iteraciones.
RF32	Generar listado de las listas de verificación.
RF33	Crear variable.

RF34	Modificar variable.
RF35	Consultar variable.
RF36	Eliminar variable.
RF37	Listar variables.
RF38	Crear escenario.
RF39	Modificar escenario.
RF40	Consultar escenario.
RF41	Eliminar escenario.
RF42	Listar evidencias.
RF43	Listar defectos y dificultades detectados.

2.5.2 Requisitos No Funcionales

"Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente a penas se aplican a características o servicios individuales del sistema." [25].

Usabilidad:

RNF 01: El sistema deberá poseer un dominio de aplicación web.

RNF 02: El sistema deberá presentar una navegación intuitiva.

Eficiencia:

RNF 03: Soportar conexiones al sistema de manera simultánea.

RNF 04: El consumo de memoria RAM deberá ser menor de 300 MB.

Restricciones de diseño e implementación:

RNF 05: Lenguaje de programación a utilizar PHP, con el uso de framework Yii, bajo el IDE NetBeans 8.0.

RNF 06: Utilizar las bibliotecas JDBC para PostgreSQL 9.2.

RNF 07: Como IDE se utilizará NetBeans en su versión 8.0 o superior.

Software:

- **Para el cliente:**

RNF 08: El sistema podrá ejecutarse sobre sistema operativo Linux Debian 6, en adelante, Ubuntu 11.10 en adelante o versiones de Window 7 o superiores.

RNF 09: Podrá utilizarse el sistema en navegadores de Mozilla, preferiblemente Firefox en su versión superior a la 24.

- **Para el servidor:**

RNF 10: El sistema podrá ejecutarse sobre sistema operativo Linux Debian 6, en adelante, Ubuntu 11.10 en adelante o versiones de Window 7 o superiores.

RNF 11: Como gestor de bases de datos se utilizará PostgreSQL en su versión 9.2.

RNF 12: Servidor con módulo PHP integrado (WAMP, XAMPP, u otros).

Hardware:

RNF 13: La capacidad requerida de la memoria RAM tiene que ser mayor o igual a 1 gigabyte.

RNF 14: El disco duro que puede soportar a la aplicación debe tener una capacidad mayor o igual a 80 gigabyte.

Requisitos de licencia

RNF 15: Licencia GPL: Netbeans versión 2 (IDE de desarrollo).

RNF 16: Licencia CDDL: Netbeans versión 2 (IDE de desarrollo).

RNF 17: Licencia PostgreSQL Licence: PostgreSQL 9.2 (Sistema de Gestión de Bases de Datos).

Seguridad:

RNF 18: Los usuarios deben autenticarse antes de interactuar con el sistema.

RNF 19: El sistema deberá garantizar el acceso controlado a la información. Este debe influir sobre cómo se presentan las interfaces para cada usuario dependiendo del nivel de acceso a la información.

2.6 Casos de Uso del Sistema

A continuación, se muestra el diagrama de caso de uso del sistema (Figura 5) donde están representados los actores que interactúan con el mismo mediante los casos de uso y luego su descripción textual.

2.6.1 Diagrama de Casos de Uso del Sistema

Un diagrama de Casos de Uso del Sistema (CUS) es una forma de diagrama de comportamiento UML mejorado. El mismo establece un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades (requisitos) que debe cumplir el sistema [26].

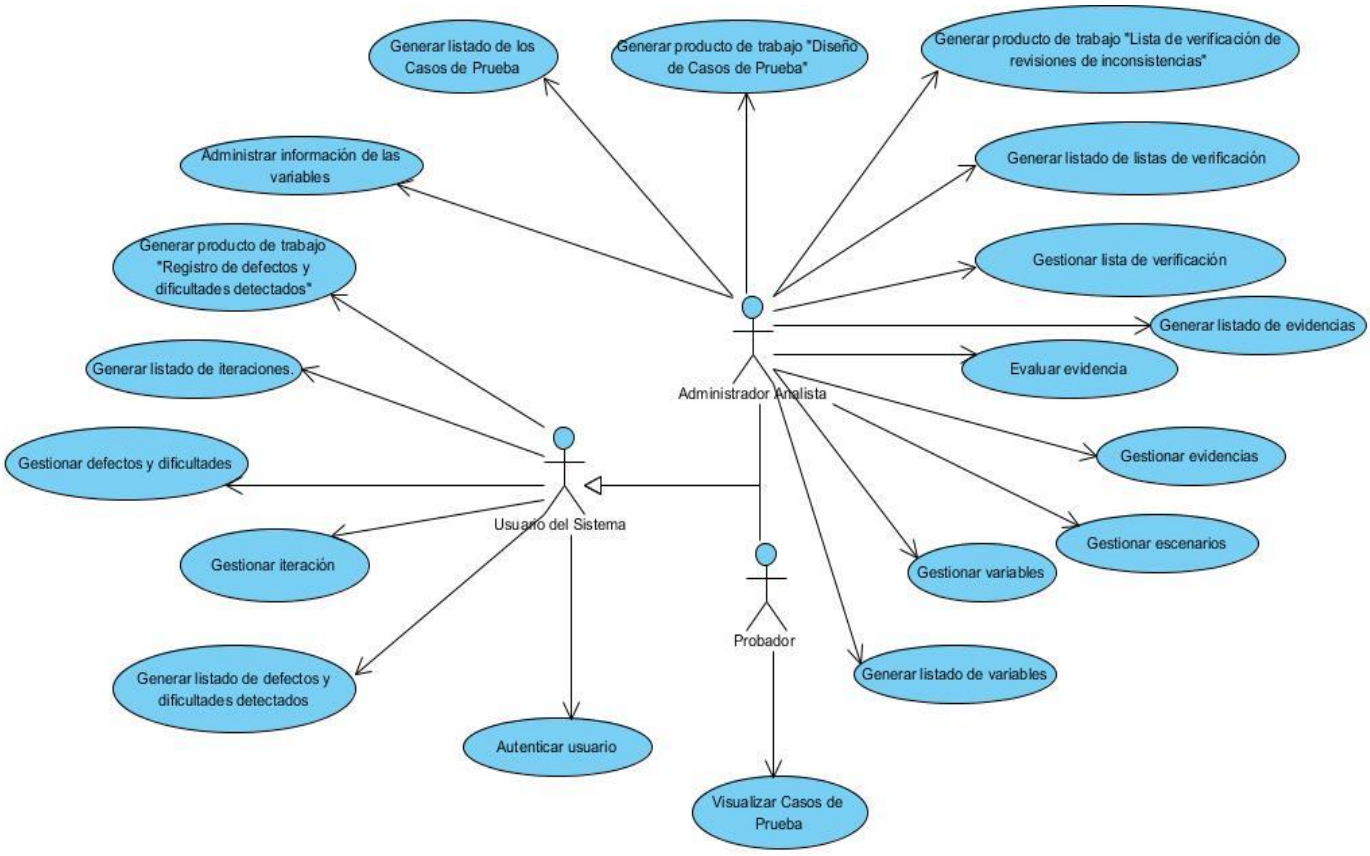


Figura 5: Diagrama de casos de uso del sistema.

2.6.2 Patrones de Casos de Uso del Sistema

Un patrón de casos de uso no describe un uso particular de un sistema, más bien se basa en la captura de técnicas para que el modelado sea mantenible, reusable, y entendible [27].

A continuación, se listan los patrones utilizados para modelar el diagrama de CUS:

- Múltiples actores-roles comunes.
- CRUD completo.

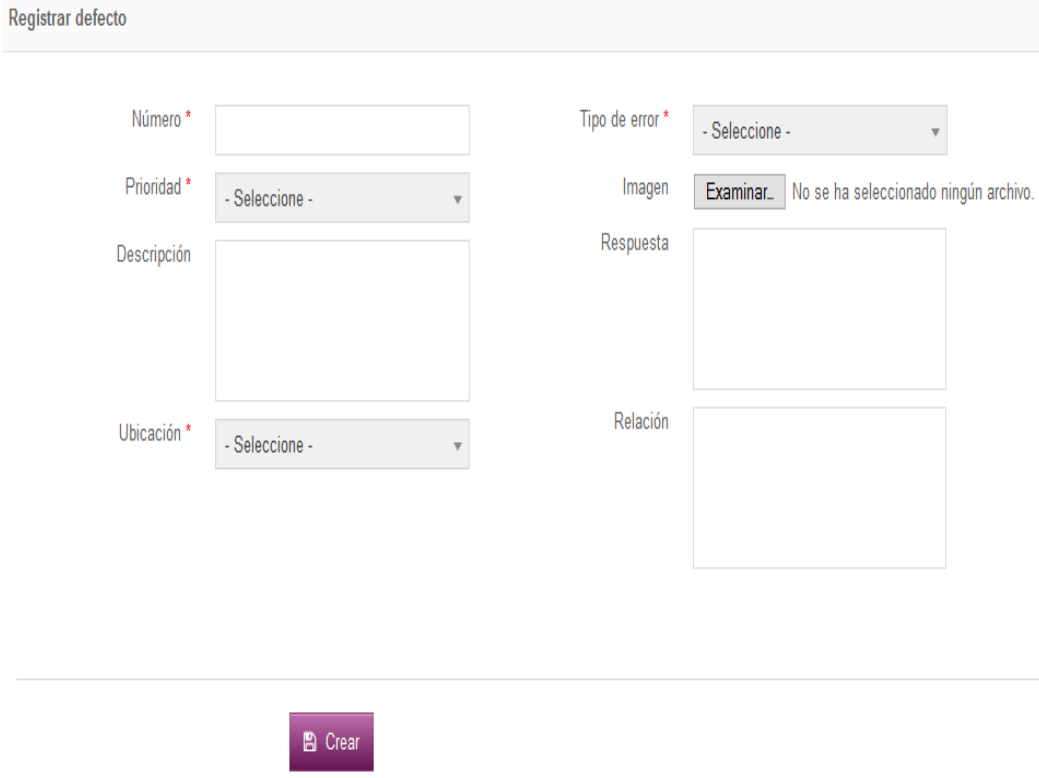
2.6.3 Descripción de los Casos de Uso del Sistema

La descripción de los casos de uso permite comprender mejor el funcionamiento de un sistema. Muestran cómo debería reaccionar el mismo ante una entrada del usuario, así como la descripción de las funcionalidades con las que cuenta.

Tabla 7: Descripción del CUS Gestionar defectos.

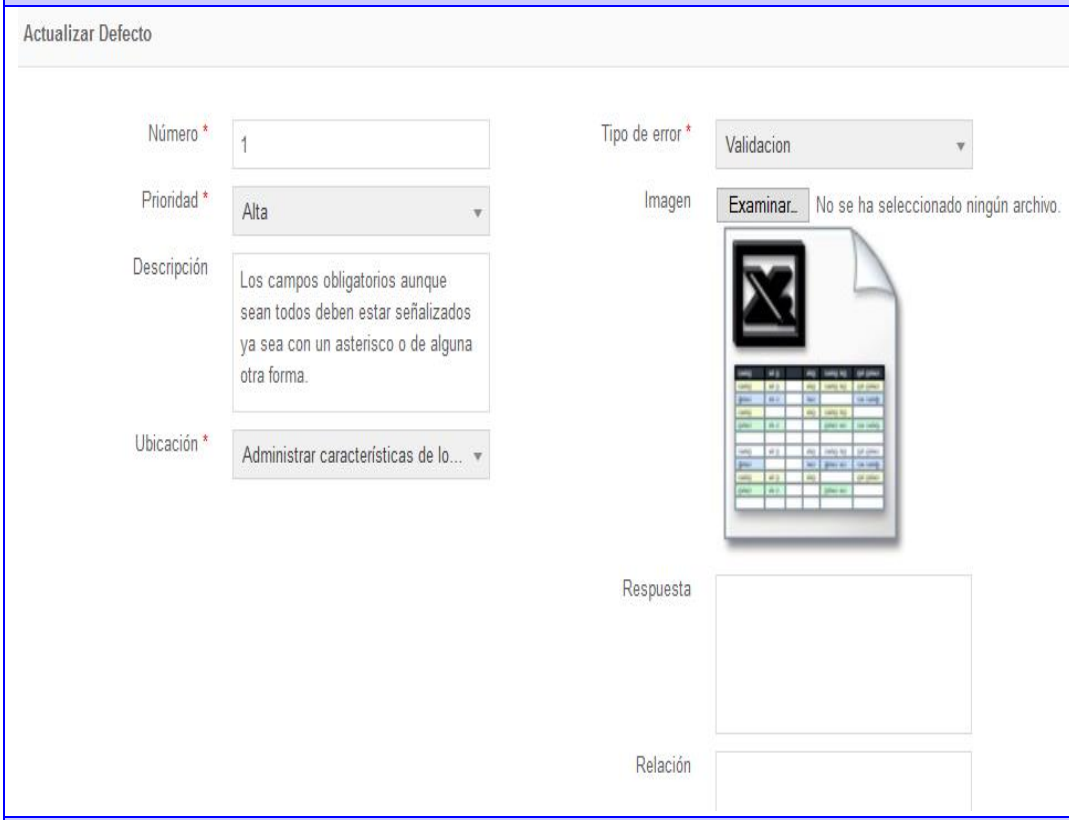
Caso de Uso	Gestionar defectos.	
Objetivo	Registrar, modificar, eliminar, o consultar registros de defectos de la base de datos del sistema.	
Actores	Administrador Analista, Probador	
Resumen	El caso de uso se inicia cuando el actor ya ha creado el producto de trabajo “Registro de defectos y dificultades”. El mismo decide registrar, modificar, eliminar o mostrar los datos del mismo.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	Actor previamente autenticado y estar vinculado a algún proyecto. Además, haber creado el producto de trabajo “Registro de defectos y dificultades”.	
Postcondiciones	Se adiciona un registro de defectos, se modifican sus datos, se elimina el mismo, o se consultan sus datos.	
Flujo de eventos		
Flujo básico Gestionar defectos		
	Actor	**Sistema**
1.	Selecciona desde la página principal la opción “Artefactos de Prueba”.	
2.		Muestra las opciones que responden a los productos de trabajo que pueden ser generados en la disciplina de Prueba: - Diseño de CP (ver CU Generar producto de trabajo “Diseño de Caso de Prueba”). - Lista de verificación (ver CU Generar producto de trabajo “Lista de verificación de evidencias”). - Registro de defectos.
3.	Selecciona la funcionalidad “Registro de defectos”.	
4.		Muestra los datos correspondientes a dicho registro: - Datos generales (ver CU Generar producto de trabajo “Registro de

		defectos y dificultades detectados"). - Control de cambios (ver sección 2 "Modificar registro" del CU Generar producto de trabajo "Registro de defectos y dificultades detectados"). - Iteración.
5.	Selecciona la pestaña iteración, para luego seleccionar el botón "Actualizar" de la iteración deseada en caso de existir.	
6.		Muestra los datos correspondientes, permitiendo realizar las siguientes operaciones: - Insertar defecto (ver Sección 1 "Insertar defecto"). - Modificar defecto (ver sección 2 "Modificar defecto"). - Eliminar defecto (ver sección 3 "Eliminar defecto"). - Mostrar defecto (ver sección 4 "Mostrar defecto").
7.		Termina el caso de uso.
Sección 1: "Insertar defecto"		
Flujo básico		
	Actor	**Sistema**
1.	Da clic en el botón "Adicionar Defecto".	
2.		Muestra una página con los campos para adicionar un defecto: - Número. - Prioridad. - Descripción. - Ubicación. - Tipo de error. - Imagen. - Respuesta. - Relación.
3.	Inserta los datos en los campos correspondientes y da clic en el botón "Crear".	
4.		Valida los datos insertados.
Prototipo funcional "Insertar defecto"		



Flujos alternos		
4a "Campos vacíos"		
1.		Muestra un mensaje en rojo con los campos que deben de ser llenados obligatoriamente.
2.	Vuelve al paso 3 de la Sección 1 "Insertar defecto".	
4b "Campos incorrectos"		
1.		Muestra un mensaje en rojo con los campos que han sido llenados incorrectamente.
2.	Vuelve al paso 3 de la Sección 1 "Insertar defecto".	
Sección 2: "Modificar defecto"		
Flujo básico		
	Actor	**Sistema**
1.	Selecciona el botón "Actualizar" del defecto a modificar.	
2.		Muestra los datos de los campos asociados al defecto.
3.	Modifica los campos que desee y presiona el botón "Actualizar".	
4.		Recarga la página.

Prototipo funcional "Modificar defecto"



Sección 3: "Eliminar defecto"

Flujo básico

	Actor	Sistema
1.	Selecciona el botón "Eliminar" del defecto deseado.	
2.		Muestra un mensaje "¿Está seguro que desea eliminar este elemento?".
3.	Selecciona el botón "Aceptar".	
4.		Recarga la página.

Flujos alternos

5a "Cancela la acción de eliminar"

1.	Presiona el botón "Cancelar".	
2.		Recarga la página.

Sección 4 "Mostrar defecto"

Flujo básico

	Actor	Sistema
1.	Selecciona el botón "Ver" del defecto deseado.	
2.		Muestra una página con los datos correspondientes al defecto.

Relaciones	**CU incluidos**	No aplica

	CU extendidos	No aplica
Requisitos no funcionales	No aplica	
Asuntos pendientes	No aplica	

Las descripciones de los CU restantes más significativos se pueden consultar en el Anexo externo: Especificación de CU.

2.7 Modelo de diseño

El modelo de diseño es planteado como un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar, constituyendo una entrada principal en la actividad de implementación [28].

2.7.1 Diagrama de clases

Los diagramas de clases (DC) son un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos; las cuales pueden ser asociativas, de herencia, de uso y de contenimiento. Expresan la estructura u organización del software en términos de clases. Además, constituyen el pilar básico del modelado con UML, siendo utilizados tanto para mostrar lo que el sistema puede hacer, como para mostrar cómo puede ser construido. A continuación, se muestran algunos diagramas de clases para los CU del sistema más significativos, teniendo en cuenta las entidades, sus atributos y relaciones. El resto de los diagramas se pueden observar en el Anexo externo “Diagramas de clases por CU”.

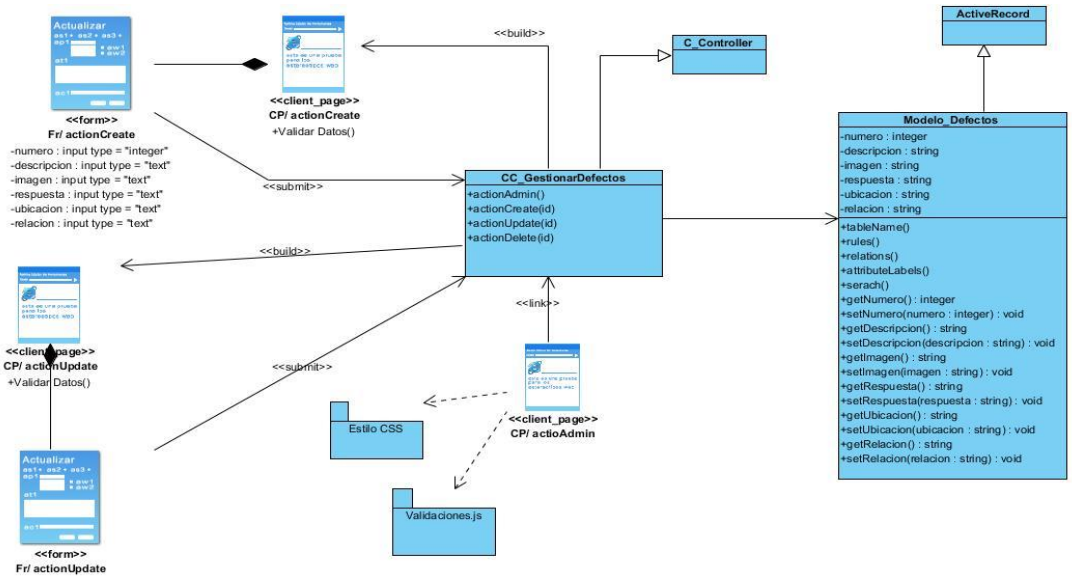


Figura 6: Diagrama de clases para el CU "Gestionar defectos".

2.7.2 Patrón Arquitectónico

Los patrones de arquitectura, o también llamados arquetipos ofrecen soluciones a problemas de arquitectura de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre como pueden ser usados [29].

Para la construcción del sistema web propuesto se define el patrón Modelo-Vista-Controlador (MVC), partiendo además de la selección de Yii como marco de trabajo el cual implementa el diseño de dicho patrón. MVC tiene como objetivo separar la lógica del negocio de las consideraciones de la interfaz del usuario para que los desarrolladores puedan modificar cada parte más fácilmente sin afectar a la otra. Por otra parte, el modelo representa la información (los datos) y las reglas de negocio; la vista contiene elementos de la interfaz de usuario, tales como texto, entradas de formulario; y el controlador gestiona la comunicación entre el modelo y la vista [30].

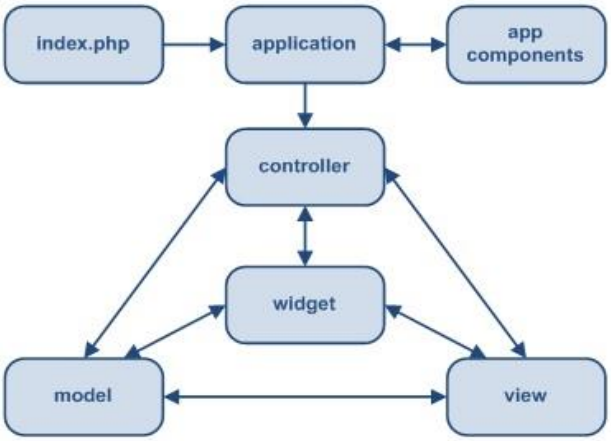


Figura 7: Patrón arquitectónico Modelo-Vista-Controlador para Framework Yii.

En el caso del framework Yii, el mismo organiza las clases según su tipo en capetas diferentes (Figura 8).

Código:
```
testweb/
   assets/
   css/
   images/
   themes/
   protected/
      commands/
         shell/
      components/
      config/
      controllers/
      data/
      extensions/
      messages/
      models/
      runtime/
      tests/
      views/
         layouts/
         site/
            pages/
```

Figura 8: Árbol de carpetas del Framework Yii.

En la carpeta *models* están todas las clases relacionadas con el modelo de la aplicación. Por otro lado, en las views están las vistas que son las interfaces de usuario. Estas se dividen en dos, las que se encuentran en la carpeta *layouts* son la parte de las vistas que no cambian (banner, módulos de botones) en las cuales demuestran los contenidos que se encuentran en la carpeta *pages*; y la *controllers* es la carpeta destinada a contener las clases controladoras que se encargan de las llamadas al modelo para obtener los datos y se los pasan a las vistas que se encuentran en pages para que los muestren al usuario. Todas las peticiones se canalizan a través de un controlador frontal llamado *application* que es el encargado de seleccionar a cuál clase controladora pasarle las peticiones del usuario (Ver Figura 8).

2.8 Diagrama de secuencia

Los diagramas de interacción son modelos que describen como grupos de objetos colaboran para conseguir algún fin. Los mismos se caracterizan por ser capaces de capturar el comportamiento de un Caso de Uso. Dichos diagramas se pueden expresar de dos maneras: diagramas de secuencia y diagramas de colaboración [31].

Un diagrama de secuencia (DS) muestra las interacciones expresadas en función de secuencias temporales; en concreto dicho diagrama muestra los objetos participantes y los mensajes que intercambian entre ellos a lo largo del tiempo. Además, son muy apropiados para especificar restricciones de interacción en tiempo real. A continuación, se representa el escenario para el CU Gestionar Defectos.

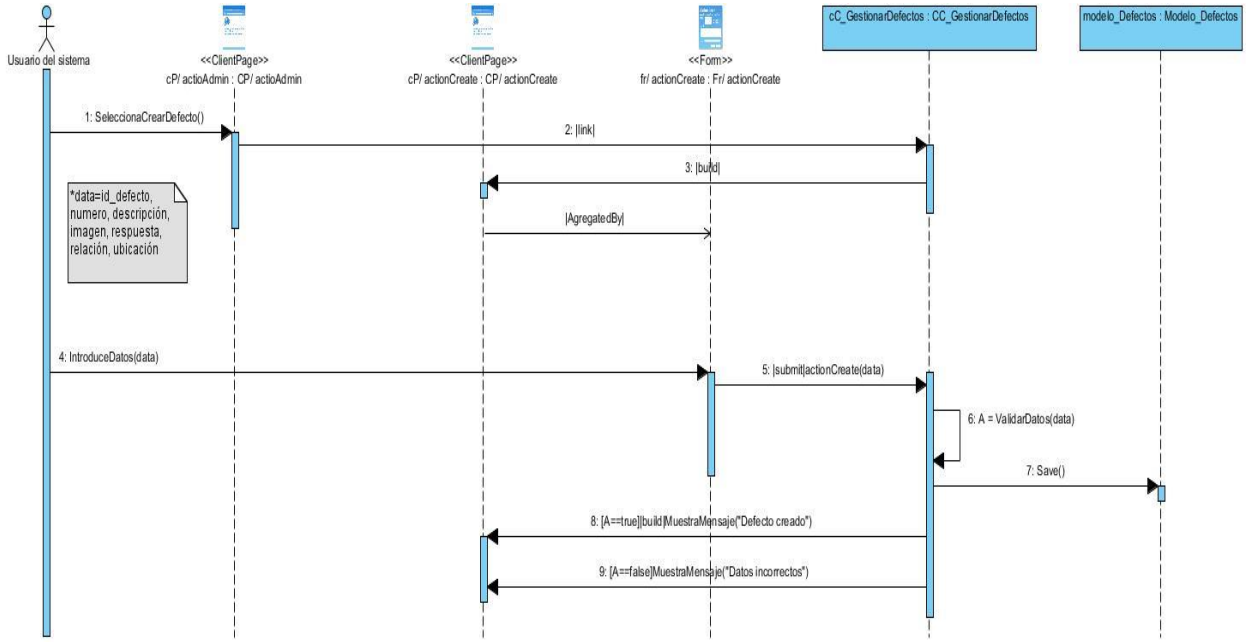


Figura 9: Diagrama de secuencia "Insertar defectos".

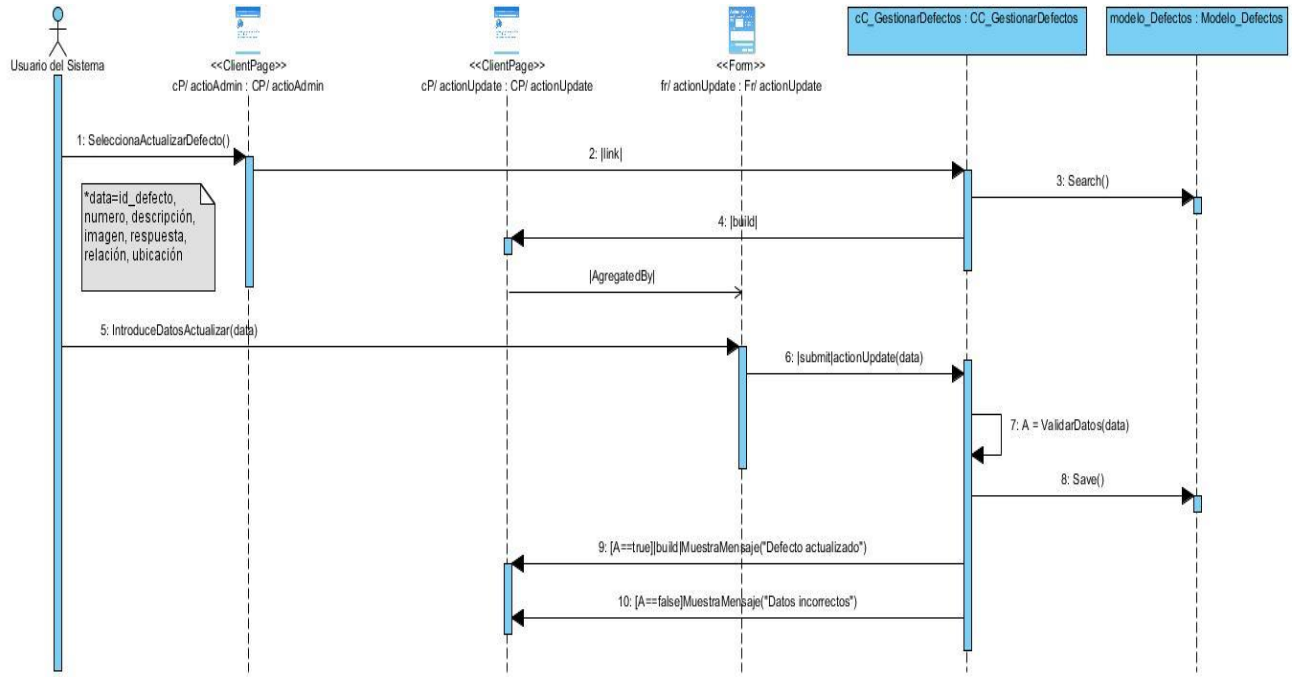


Figura 10: Diagrama de secuencia "Modificar defectos".

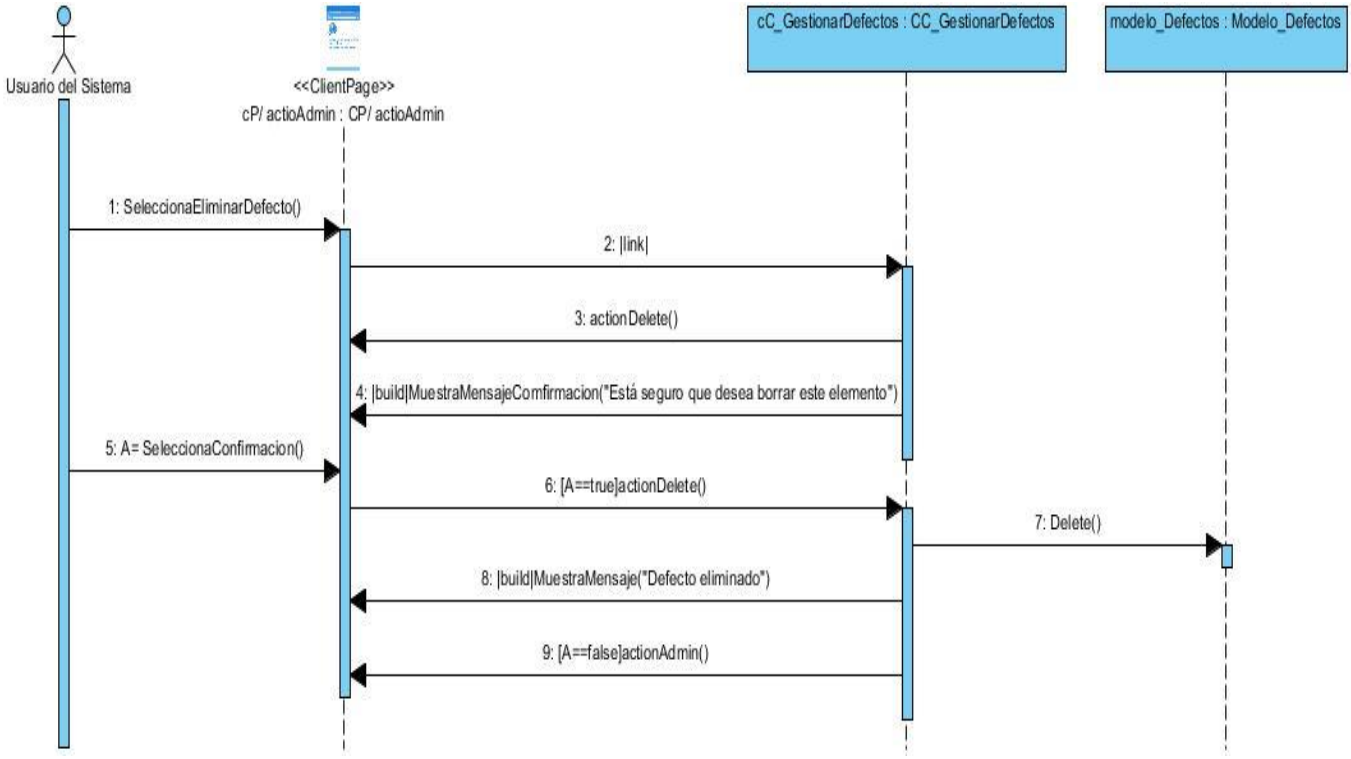


Figura 11: Diagrama de secuencia "Eliminar defectos".

Para consultar los restantes diagramas de secuencia puede dirigirse al Anexo externo “Diagramas de secuencia”.

2.9 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Contribuyen a reutilizar diseño gráfico identificando aspectos claves de la estructura de un diseño que puede ser aplicado en una gran cantidad de situaciones [29].

Para la implementación de la propuesta de solución fueron aplicados los siguientes patrones de diseño:

“***G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns* (GRASP), son una serie de patrones que describen los principios fundamentales de la asignación de responsabilidades a objetos y son considerados una serie de buenas prácticas en el diseño de software” [32].

El patrón Creador ayuda a identificar quien debe ser el responsable de la creación o instanciación de nuevos objetos o clases. Tiene la información necesaria para realizar la creación del objeto que es una de las actividades más comunes en un sistema orientado a objetos. Almacena o maneja varias instancias de la clase. En la siguiente figura (ver Figura 12) se muestra el uso de dicho patrón.

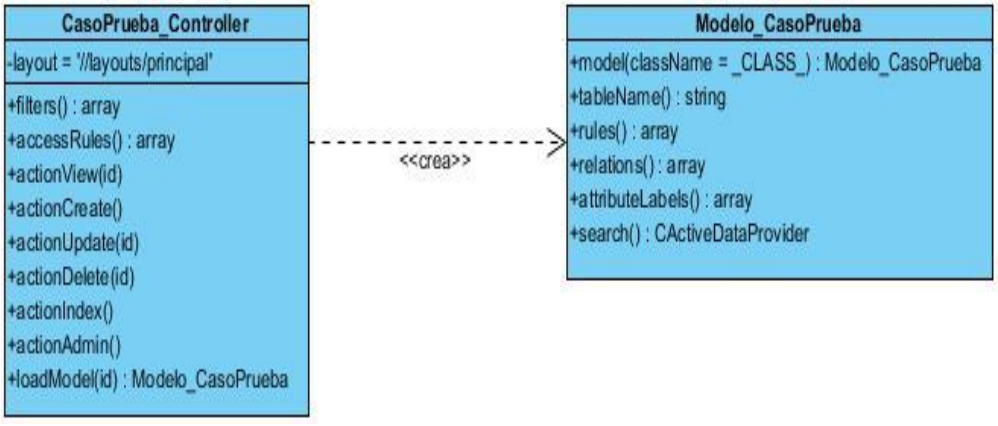


Figura 12: Diagrama de clases que representa el Patrón Creador para CUS “Gestionar Caso de Prueba”.

El patrón Controlador asigna la responsabilidad de gestionar un mensaje de un evento del sistema a una clase controladora. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. En la siguiente figura (ver Figura 13) se muestra el uso de dicho patrón.

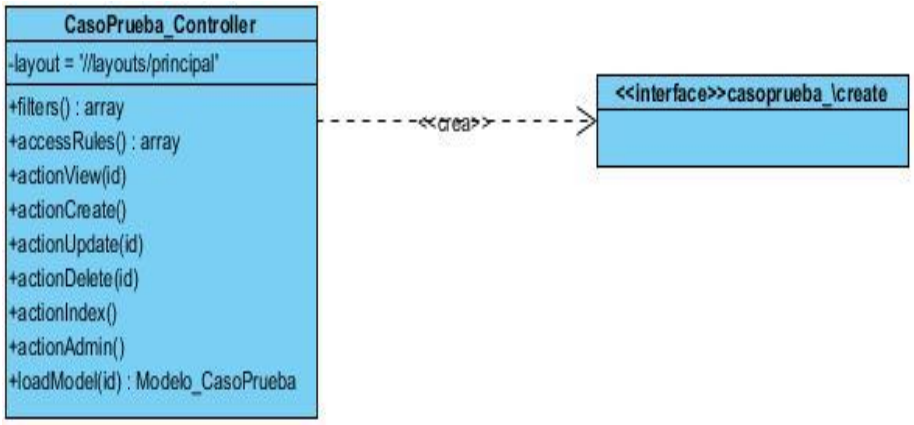


Figura 13: Diagrama de clases que representa el Patrón Controlador para CUS "Gestionar Caso de Prueba".

Además, es aplicado el patrón Bajo Acoplamiento. El mismo establece tener pocos componentes que dependan de otros, pues mientras menos acoplamiento haya más reusable y flexible se vuelve el sistema. En la aplicación, las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, proporcionando esto que la dependencia en este caso sea baja.

Se evidencia también el uso del patrón Alta Cohesión el cual plantea que la información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase. Para la implementación de la aplicación se hizo uso de las formas de organización que brinda Yii, pues permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión.

"***G**ang-**o**f-**F**our* (GoF), también conocidos como 'la pandilla de los cuatro' son patrones de diseño utilizados en situaciones muy frecuentes debido a que se basan en la experiencia acumulada al resolver problemas reiterativos. Además, favorecen la reutilización del código." [33].

A continuación, se detallan algunos de los patrones GoF que se utilizaron en el diseño:

Singleton: Este patrón propone una sola instancia de la clase controladora para acceder a los datos y modificarlos, al igual que el *framework* Yii, que usa una sola instancia para manejar las sesiones y los *layout*.

Decorator: Permite añadir funcionalidades dinámicamente a una clase con la creación de otra clase, sin necesidad de la herencia. En el *framework* Yii, los *layouts* decoran a las vistas, pues en un momento determinado les confieren propiedades dinámicas que no poseían.

Observer: Define una dependencia de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes. Se trata de un patrón de comportamiento, es decir, está relacionado con algoritmos de funcionamiento y asignación de responsabilidades a clases y objetos. Se evidencia en la implementación del patrón arquitectónico MVC definido en la aplicación.

2.10 Modelo de datos

Una base de datos sirve para almacenar la información que se utiliza en un sistema de información determinado. Las necesidades y los requisitos de futuros usuarios del sistema en cuestión se deben tener en cuenta a la hora de realizar el modelado de la base de datos correspondiente.

El modelo entidad-relación es uno de los enfoques de modelización de datos más utilizado debido a su simplicidad y legibilidad, la cual se ve favorecida ya que proporciona una notación diagramática muy comprensiva [34].

El modelo de datos de la aplicación posee una alta complejidad, debido a que se utiliza la extensión CRUGE[3] de Yii, la cual genera un número considerable de entidades manejadas por el sistema, por lo que no se contemplarán todas en el modelo, en su lugar solo se contemplarán aquellas entidades principales que conforman el sistema base (ver Figura 14: Muestra del diagrama Entidad-Relación de la base de datos).

2.11 Conclusiones parciales

Tras la definición de las características del sistema se arrojan las siguientes conclusiones:

- La definición de los requerimientos de la aplicación permitió identificar los requerimientos con los que contará el sistema propuesto, los cuales darán respuesta a las necesidades del problema.
- Se generaron los productos de trabajo que describen la solución propuesta acorde a lo que propone la metodología de desarrollo utilizada.
- Se seleccionaron los patrones de arquitectura y diseño para garantizar el correcto funcionamiento y organización del sistema.

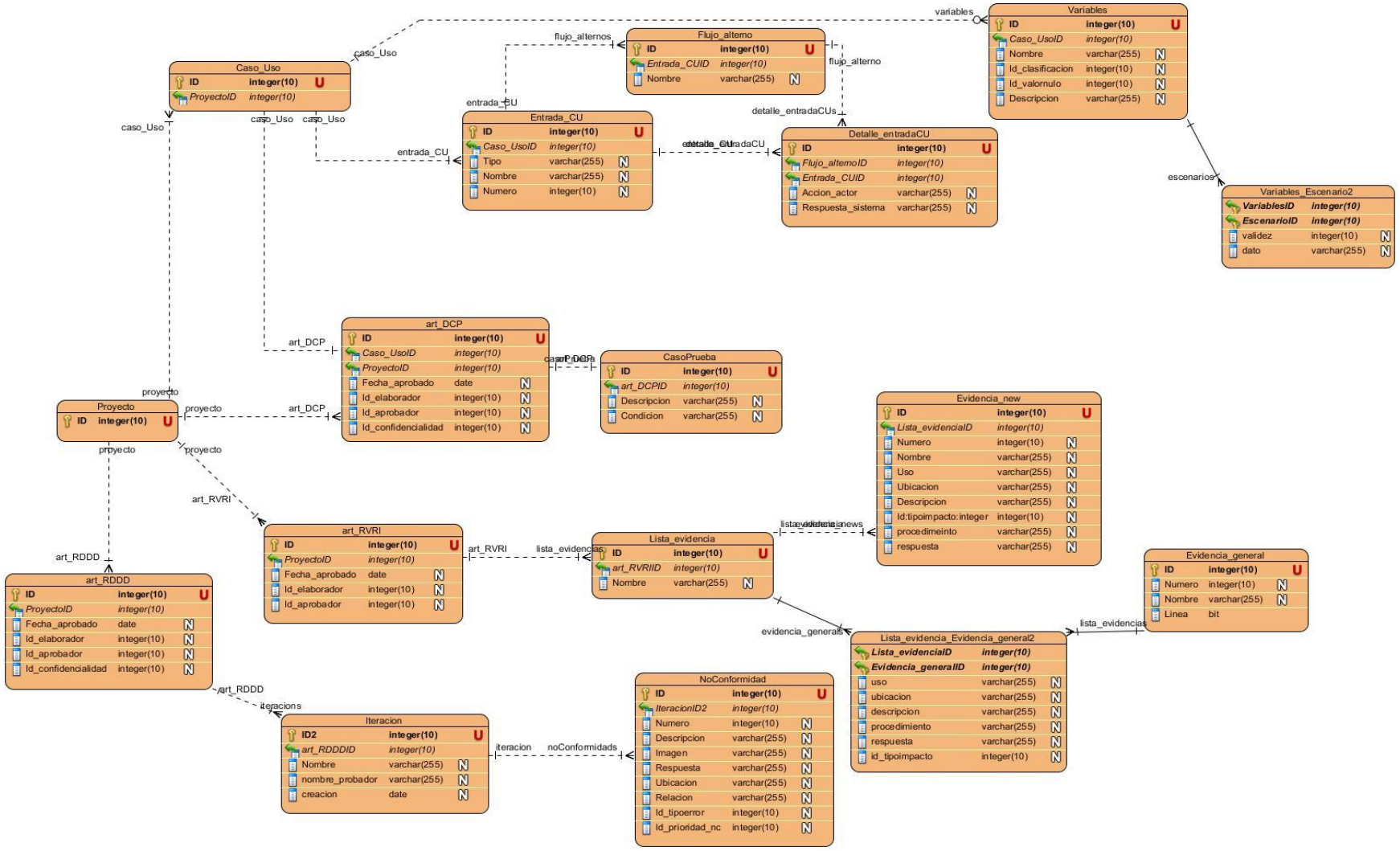


Figura 14: Muestra del diagrama Entidad-Relación.

Capítulo #3. Implementación y pruebas del sistema

3.1 Introducción

Partiendo del resultado del análisis y diseño, en el presente capítulo se generan los artefactos referentes a las disciplinas de implementación y prueba del software. Como principales modelos se definen los diagramas de componentes y de despliegue. Además, se diseñan los casos de pruebas pertinentes y por último se realizan una serie de pruebas a la propuesta de solución con el objetivo de justificar su valía.

3.2 Estándar de codificación

El estándar de codificación define la estructura y apariencia física del código, lo que facilita su comprensión, mantenimiento y lectura. En la implementación se utilizaron diferentes estilos que se describen a continuación:

3.2.1 Definición de clases

El nombre de las clases está escrito en inglés-español y su llave de apertura se encuentra en la próxima línea del nombre (ver Figura 15).

```
class ArtNcController extends Controller
{

        public $defaultAction = 'admin';


        public function filters()
        {
            return array_merge (
                array(
                    'accessControl',
                    array('CrugeAccessControlFilter')
                )
            );
        }
        /**
```

Figura 15: Definición de clases.

3.2.2 Declaración de variables

Las variables se escriben con minúscula y serán antecedidas por el signo ($). El signo igual (=) estará ubicado entre espacios (ver Figura 16).

```
$model = ArtNc::model()->findByPk($id);

$temp = Iteracion::model()->findByAttributes(array('id_artnc'=>$id));

$temp1 = NoComformidades::model()->findAllByAttributes(array('id_iteracion'=>$temp->primaryKey));
```

Figura 16: Declaración de variables.

3.2.3 Definición de métodos

Los métodos tienen inicialmente la palabra reservada **public**, seguida de otra palabra reservada **function**. El nombre del método tiene letra inicial minúscula, en caso de ser dos palabras se mantendrán unidas y la segunda comienza con mayúscula. En caso de pasarle elementos por parámetros, estos se escriben con minúscula (ver Figura 17).

```
public function actionView($id)
{
        $this->render('view',array(
                'model'=>$this->loadModel($id),
        ));
}
```

Figura 17: Definición de métodos.

3.2.4 Estructuras de control

Dentro de las estructuras de control se pueden encontrar *if, for, while, do while*, entre otras. Se utilizan, fundamentalmente*: if, elseif, for* y *foreach.*

```
<?php if(count($iteraciones)>0){
    for ($r=0; $r< count($iteraciones);$r++) {?>
<tr>
  <td><?php echo $iteraciones[$r]->nombre_iteracion; ?></td>
  <td><?php echo $iteraciones[$r]->nombre_probador; ?></td>
  <td><?php echo $iteraciones[$r]->creacion; ?></td>
  <td style="width: 60px;">
      <a  class="btn btn-default btn-grig-mini btn-info light"   title="Ver" ><i class=
      <a  class="btn btn-default btn-grig-mini btn-success light" href="<?php echo Yii:
      <a  class="btn btn-default btn-grig-mini btn-danger light desIteracion" id="des_<
  </td>
</tr>
    <?php  } }?>
```

Figura 18: Estructuras de control.

3.3 Diagrama de Componentes

Los diagramas de componentes permiten ver el modelado de un sistema o subsistema. Permite especificar un componente con interfaces bien definidas. Además, permite también visualizar la estructura de alto nivel del sistema y el comportamiento del servicio que estos componentes proporcionan y usan a través de interfaces [35]. La Figura 19 muestra el diagrama de componentes del CU Gestionar defectos.

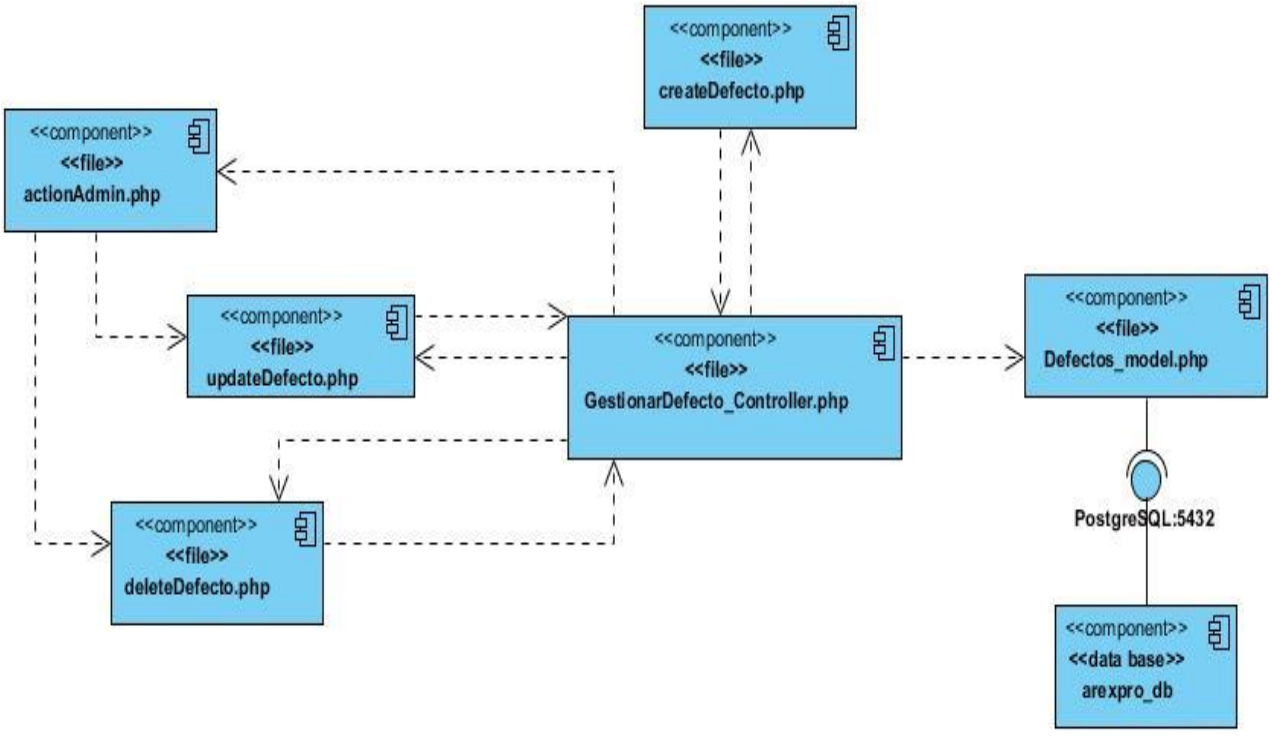


Figura 19: Diagrama de Componentes del CU Gestionar Defectos.

Los restantes diagramas de componentes se pueden consultar en el Anexo externo “Diagrama de componentes”.

3.4 Diagrama de Despliegue

“Los diagramas de despliegue son los complementos de los diagramas de componentes que, unidos, proveen la vista de implementación del sistema. Describen la topología del sistema, la estructura de los elementos de hardware y software que ejecuta cada uno de ellos. Muestran la configuración en funcionamiento del sistema. Además, muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos.” [36].

A continuación, se muestra el diagrama de despliegue que se corresponde con el módulo que se desea implementar (ver Figura 20).

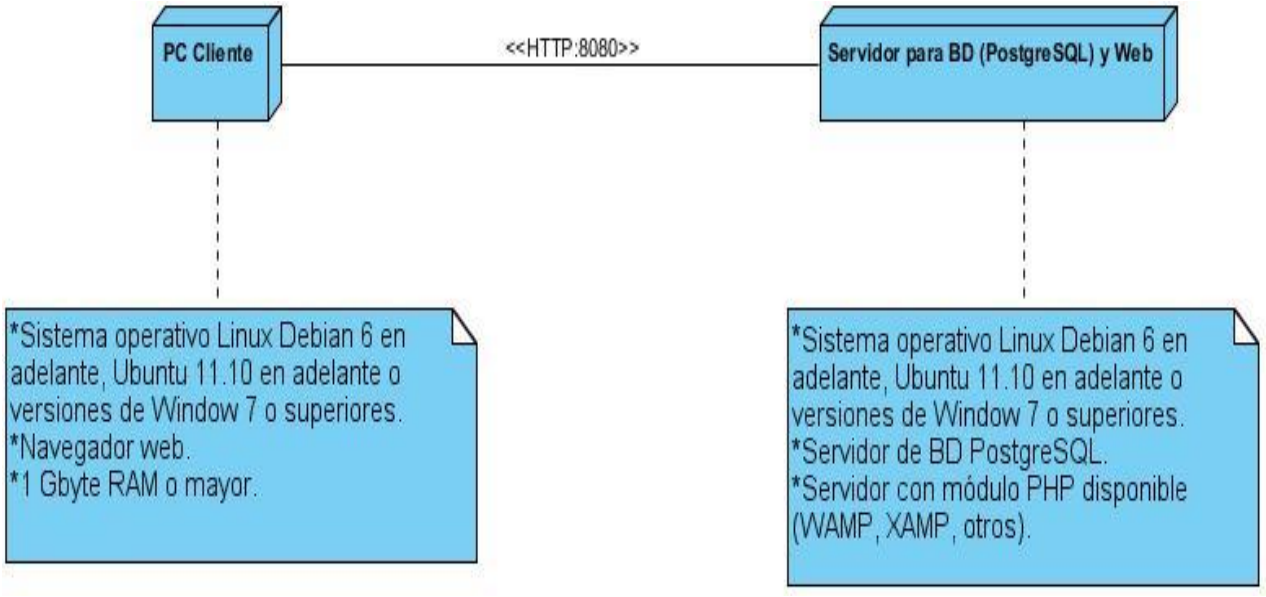


Figura 20: Diagrama de Despliegue.

3.5 Pruebas

“Las pruebas del software son el proceso que permite verificar, garantizar y mostrar la calidad de un producto, a través de resultados registrables que proporcionan una evaluación y representa una revisión final de las especificaciones, del diseño y de la codificación. Son empleadas para identificar posibles fallos durante el proceso de desarrollo. Los casos de prueba son actividades en las cuales un sistema o componente es ejecutado bajo condiciones o requerimientos especificados, permitiendo encontrar y documentar los defectos que puedan afectar la calidad del software” [37].

3.5.1 Diseño de Casos de Prueba

Los casos de prueba son la forma de verificar las diversas funcionalidades existentes en un producto de software descritas en el formato de los Casos de Uso. Estos se realizan con el objetivo de conseguir un margen de confianza aceptable, de que serán encontrados todos los defectos existentes sin consumir una cantidad excesiva de recursos.

A continuación, se relacionan los Casos de Prueba para los Casos de Uso especificados (ver Anexo externo “Casos de Prueba”):

- Caso de Pruebas del Caso de Uso Gestionar defectos.
- Caso de Pruebas del Caso de Uso Gestionar Casos de Pruebas.
- Caso de Pruebas del Caso de Uso Gestionar evidencias.
- Caso de Pruebas del Caso de Uso Evaluar evidencias.
- Caso de Pruebas del Caso de Uso Gestionar defectos.
- Caso de Pruebas del Caso de Uso Generar producto de trabajo “Diseño de Casos de Prueba”.
- Caso de Pruebas del Caso de Uso Generar producto de trabajo “Lista de verificación de revisión de inconsistencias”.
- Caso de Pruebas del Caso de Uso Generar producto de trabajo “Registro de defectos y dificultades detectados”.
- Caso de Pruebas del Caso de Uso Administrar información de las variables.
- Caso de Pruebas del Caso de Uso Gestionar iteración.
- Caso de Pruebas del Caso de Uso Gestionar lista de verificación.
- Caso de Pruebas del Caso de Uso Gestionar variables.

3.5.2 Resultados de las pruebas realizadas

Teniendo en cuenta las condiciones de la solución informática propuesta para la gestión y administración de los productos de trabajo asociados a la disciplina de Pruebas, se realizan diversas pruebas de las cuales se exponen sus resultados:

Pruebas de funcionalidad:

Dentro de las técnicas empleadas para las pruebas de caja negra se utiliza, partición de equivalencia con el objetivo de examinar los valores válidos e inválidos de las entradas existentes en la aplicación (ver Figura 22).

Pruebas de rendimiento:

Haciendo uso de la herramienta Apache JMeter se aplican pruebas de rendimiento a diferentes funcionalidades del sistema arrojando los siguientes resultados:

Tabla 8: Resultados de las pruebas de rendimiento.

Funcionalidad	Usuarios (Muestras)	Tiempo de ejecución (ms)			Rendimiento		
		Mín.	Max.	Media	% Error	Pet./seg	Kb./seg
Generar producto de trabajo "Registro de defectos y dificultades detectados".	50	127	3361	1493	0.0%	27.4/sec	446.3
	100	307	6907	3164	0.0%	27.5/sec	446.4
Gestionar evidencia.	50	3	3209	1452	0.0%	29.8/sec	210.6
	100	6	13508	2582	0.0%	30.4/sec	215.4
Agregar defecto.	50	191	4024	1535	0.0%	27.2/sec	222.6
	100	358	11319	3691	0.0%	22.9/sec	187.2
Describir variables.	50	594	4167	2402	0.0%	18.3/sec	151.7
	100	581	10755	4848	0.0%	18.9/sec	156.5
Evaluar evidencia.	50	241	5145	2165	0.0%	18.1/sec	152.6
	100	438	16067	4529	0.0%	18.2/sec	153.7
Generar producto de trabajo "Lista de verificación de revisión de inconsistencias".	50	135	5217	1611	0.0%	25.9/sec	378.3
	100	144	6972	2979	0.0%	29.0/sec	423.1
Generar producto de trabajo "Diseño de Caso de Prueba".	50	218	4575	1593	0.0%	25.0/sec	421.3
	100	285	5810	3378	0.0%	24.6/sec	414.0

- Mínimo (Mín.): Indica el mínimo de tiempo de ejecución invertido para una petición con n (columna Muestras) usuarios haciendo peticiones de manera concurrente.
- Máximo (Máx.): Indica el máximo de tiempo de ejecución invertido para una petición con n (columna Muestras) usuarios haciendo peticiones de manera concurrente.
- Media: Representa el tiempo de ejecución promedio de una petición con n usuarios.
- Error: Indica la relación entre el total de peticiones y el número de peticiones que originaron errores

- Rendimiento (Pet./seg): Hace referencia al número de peticiones que el servidor puede procesar en un segundo.
- Rendimiento (Kb./seg): Hace referencia a la cantidad de datos que el servidor puede procesar en un segundo

Pruebas de integración:

Lograr evaluar una variable en un caso de prueba depende de los resultados arrojados por el sistema AREXPRO al realizar una consulta SQL. Dicha consulta se encarga de capturar los nombres de las diferentes variables asociadas a la especificación de Caso Uso en cuestión, para la cual se ha confeccionado un CP. Al realizar la integración del módulo con el sistema AREXPRO, se pudo observar que la funcionalidad “Update” perteneciente a la clase “ArtDCasoPrueba” arroja resultados satisfactorios. Puesto que se puede comprobar que las variables especificadas para el CP son las mismas que fueron definidas con anterioridad en la especificación de Caso de Uso. De igual manera se realiza un análisis similar a la misma funcionalidad, pero teniendo en cuenta otros aspectos como las secciones, las cuales son obtenidas también mediante una consulta SQL. Los resultados obtenidos con la aplicación de estas pruebas también se encuentran reflejados en la gráfica que se presenta en la Figura 22.

Pruebas de aceptación:

Se planifica un encuentro con algunos miembros de proyectos de desarrollo del centro VERTEX y del Centro de Tecnologías para la Formación (FORTES), los cuales se han desempeñado en los roles de la gestión de proyectos, ya sea como analista, administrador de calidad o probadores. Luego de explicado el funcionamiento y los objetivos del sistema se les dio permiso para acceder al mismo. Dentro de este grupo estuvieron presentes 6 especialistas (Saylin Salas Hechavarría, Yeili Ibarra Monteagudo, Lisleidy Mier Pierre, Lizandra Hernández Hernández, Linet Katiuska Remón Salcedo, Sandy Guerra Fernández), de los cuales 4 mostraron un nivel de satisfacción alto, 1 medio, y 1 no estuvo conforme con la propuesta de solución (ver Figura 21), de acuerdo a las respuestas de las preguntas realizadas (ver Anexo “Encuesta de satisfacción”).

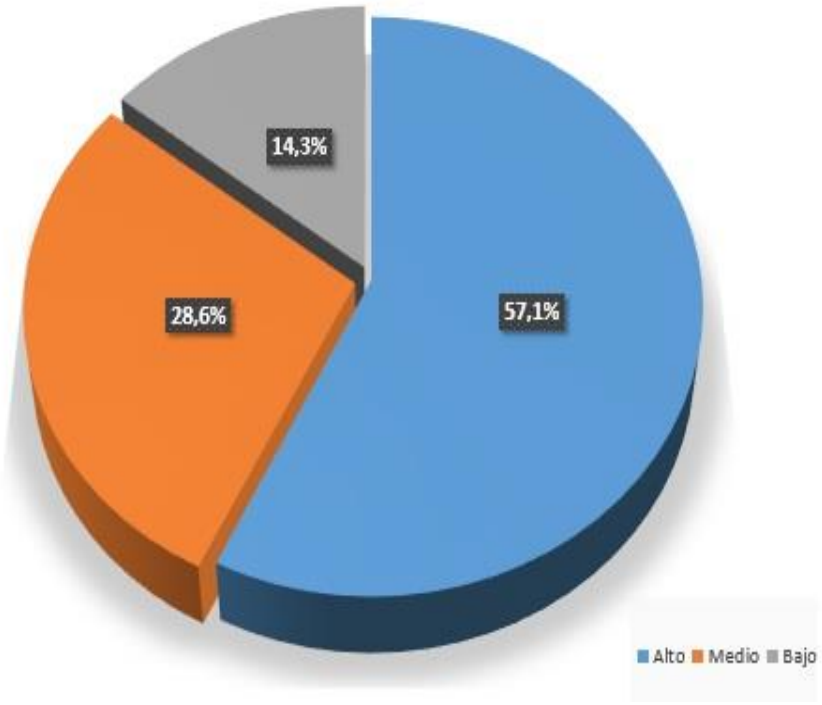


Figura 21: Nivel de aceptación.

Una vez diseñados los casos de prueba, se procedió a la ejecución de las pruebas. Los resultados se pueden observar en la Figura 22, donde se muestran la cantidad de no conformidades encontradas, las cuales fueron corregidas. Se realizaron 4 iteraciones de pruebas, de las cuales las tres primeras resultaron con deficiencias y para la cuarta se erradicaron las mismas. En la primera se detectaron 43 defectos, en la segunda 26, en la tercera 11 y en la cuarta no se detectó ninguna.

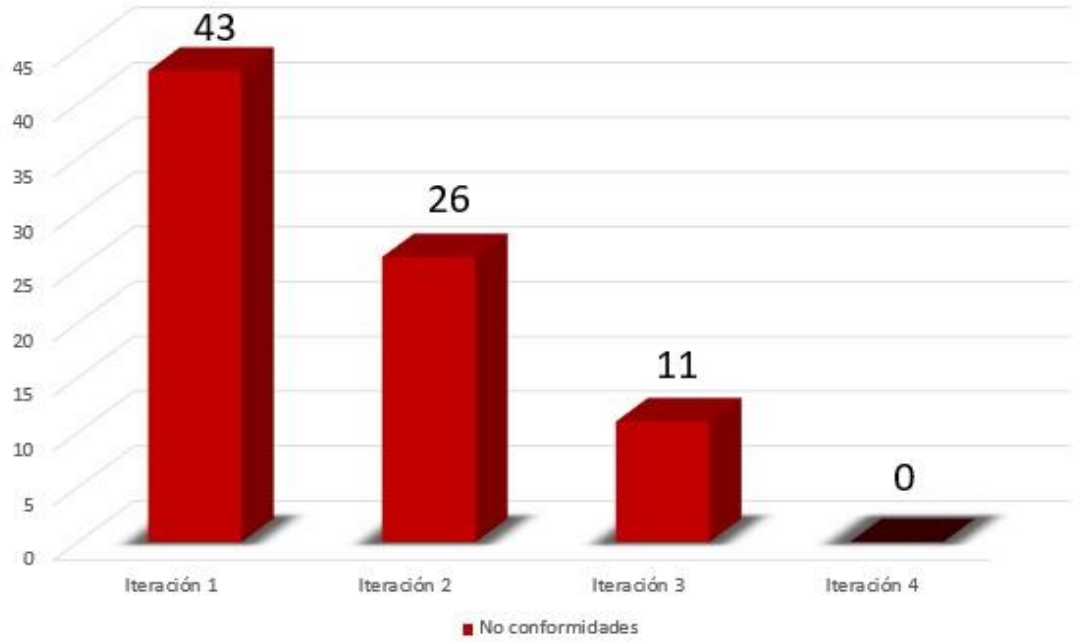


Figura 22: No conformidades por iteración.

3.6 Conclusiones parciales

Tras definir las características de implementación y pruebas de la propuesta de solución, se plantean las siguientes conclusiones:

- Se explicó el estándar de codificación, declaración de clases, variables y métodos, para lograr un mejor entendimiento del código resultante.
- Se realizaron los diagramas de componentes y despliegue para el modelado de la estructura general del sistema y de una topología de hardware donde se ejecuta el mismo.
- Se diseñaron los casos de prueba para rectificar los defectos presentes en el sistema.
- Se validó la solución propuesta a través de las pruebas de funcionalidad, rendimiento, aceptación e integración, las que permitieron evaluar el funcionamiento del sistema.

Conclusiones generales

Con el desarrollo del presente trabajo se obtuvo finalmente un módulo de apoyo para la generación de productos de trabajo en la disciplina de Prueba en escenarios de desarrollo del Centro de Entornos Interactivos 3D, para lo cual:

- La utilización de las herramientas y tecnologías: NetBeans 8.0, PHP 5.2.0, PostgreSQL y *framework* Yii, permitieron el desarrollo de la solución informática obtenida garantizando la integración de esta con el sistema de generación de Artefactos del Expediente de Proyecto (AREXPRO).
- La representación de los diagramas: modelo conceptual, caso de uso del sistema, clases del diseño, secuencia y modelo de datos, permitieron establecer un punto de partida para el desarrollo de la solución de forma estructurada, siguiendo debidamente las pautas establecidas por la metodología AUP para el proceso de desarrollo de software.
- Con la implementación de la solución, se logró obtener un módulo de apoyo capaz de gestionar y generar los productos de trabajo asociados a la disciplina de Pruebas definidos en el expediente de proyectos de desarrollo en su versión 4.0.
- La realización de los tipos de pruebas de funcionalidad, rendimiento y los niveles de pruebas de aceptación e integración, permitieron detectar los posibles defectos antes de desplegar la solución informática. Además, esto demostró el cumplimiento adecuado de las especificaciones realizadas y la obtención de un producto final con calidad.

Recomendaciones

Una vez culminada la investigación y teniendo en cuenta las experiencias obtenidas en el desarrollo de la misma, se proponen las siguientes recomendaciones:

- Incluir un módulo de trazas capaz de autogenerar el control de cambios de los artefactos del expediente de proyecto y muestre estadísticas del comportamiento de variables asociadas a la información gestionada en el sistema a petición del usuario.
- Extender el uso del sistema a todos los proyectos de la universidad.

Referencias bibliográficas

[1] N. Juristo, A. M. Moreno y S. Vegas, Técnicas de evaluación de software., 2006.

[2] L. F. Scalone, Maestría en Ingeniería en Calidad, 2006, pp. 25-230.

[3] Dirección Calidad UCI, Propuesta de Sistema de Gestión de la Calidad, 2012.

[4] «Calidad del Software,» 2016. [En línea]. Available: http://dankocs2012.blogspot.com/2012/11/modelos-y-estandares-de-calidad-del.html. [Último acceso: 14 mayo 2015].

[5] Eleven Path, «Eleven Path,» [En línea]. Available: http://blog.elevenpaths.com/2014/09/qa-pruebas-para-asegurar-la-calidad-del.html. [Último acceso: 15 06 2016].

[6] J. TIAN, Software Quality Engineering. IEEE Computer Society Executive Staff, 2005.

[7] International Software Testing Qualifications Board, Probador Certificado. Programa de estudio de nivel básico., 2010.

[8] Universidad de las Ciencias Informáticas, «Mejora de Procesos de Software,» [En línea]. Available: http://mejoras.prod.uci.cu/. [Último acceso: 23 10 2015].

[9] Junta de Andalucia, «Portal de la Junta de Andalucía,» [En línea]. Available: http://www.juntadeandalucia.es/seervicios/madeja/contenido/recurso/388. [Último acceso: 04 02 2016].

[10] TestingBaires, «TestLink 1.9.6. Última versión para bajar 2013-03-09,» 2013.

[11] autores, G.d. Metodologías de desarrollo, 2006.

[12] Alpizar Naranjo, D.e.A.O y Iván, El proceso Unificado Ágil v1.1, 2006.

[13] S. Ambler W, AUP Phases. Universidad Nacional de Costa Rica, 2006.

[14] I. JACOBSON y G. BOOCH, et al. El Lenguaje Unificado De Modelado (UML). Disponible en: JACOBSON, I.; BOOCH, G., et al. El Lenguaje Unificado de Modelado (UML) [Consultado et al. El Lenguaje Unificado de Modelado(UML), 1999.

[15] A. Comas, «JAVA o PHP,» 2004.

[16] C. B. E. AGUIRRE y P. ESQUIVEL, Comparativa de Framewok para el desarrollo de aplicaciones con php., 2012.

[17] Yii framework, «Yii framework,» [En línea]. Available: http://www.yiiframework.com/doc/guide/1.1/es/quickstart.what-is-yii. [Último acceso: 13 06 2016].

[18] POSTGRESQL, PostgreSQL Global Development Group, 2012.

[19] M. ROUSE, Integrated development environment (IDE) definition, 2009.

[20] Z. STUDIO, The PHP IDE for Smarter Development | Zend Studio, 2014.

[21] S. MICROSYSTEMS, NetBeans IDE 8.0, 2014.

[22] JETBRAINS, PHP IDE :: JetBrains PhpStorm, 2014.

[23] PHPExcel, «PHPExcel,» [En línea]. Available: http://phpexcel.codeplex.com/. [Último acceso: 13 06 2016].

[24] C. Larman y . P. Hall, «UML y Patrones.2ª Edición.,» 2013. [En línea]. Available: http://www.ceneinnova.com/eddyesanchez/archivos/ads/UML%20y%20Patrones%20%202da%20Edicion.pdf.

[25] I. Sommerville, Ingeniería de software, Séptima edición ed., Madrid: PEARSON EDUCACIÓN. S.A., 2005.

[26] Microsoft, «Developer Network,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/dd409432.aspx. [Último acceso: 15 06 2016].

[27] SG BUZZ, [En línea]. Available: http://sg.com.mx/content/view/510. [Último acceso: 25 02 2016].

[28] J. A. López Martínez, «Sistema para la gestión de Oportunidades de Negocio para el Centro de Informática Industrial,» 2015.

[29] WordPress, «Ingenieria del Software,» [En línea]. Available: https://arlethparedes.wordpress.com/2012/08/27/patrones-de-arquitectura-vs-patrones-de-diseno/. [Último acceso: 15 06 2016].

[30] YiiFramework, [En línea]. Available: http://www.yiiframework.com/doc/guide/1.1/es/basics.mvc. [Último acceso: 23 02 2016].

[31] [En línea]. Available: http://www.infor.uva.es/~mlaguna/cd/CD4.PDF. [Último acceso: 28 febrero 2016].

[32] «Prácticas de software,» [En línea]. Available: https://www.practicasdesoftware.com.ar/2011/03/patrones_grasp. [Último acceso: 24 02 2016].

[33] EcuRed, [En línea]. Available: http://www.ecured.cu/Patrones_Gof. [Último acceso: 25 02 2016].

[34] D. Costal Costa, Introducción al diseño de bases de datos., 2015.

[35] SlideShare, [En línea]. Available: http://es.slideshare.net/uitron/diagrama-de-componentes-7551535. [Último acceso: 15 06 2016].

[36] Ecured, «Diagrama de despliegue,» [En línea]. Available: http://www.ecured.cu/Diagrama_de_despliegue. [Último acceso: 05 04 2016].

[37] Pressman, Ingeniería de Software. s.l. : McGraw-Hill Interamericana de España, 2010.

Anexos

Encuesta de satisfacción

Modelo de encuesta aplicado a los 7 trabajadores implicados en el proceso de validación de la propuesta de solución, con el objetivo de medir el nivel de satisfacción con el uso del sistema informático para el apoyo al trabajo ingenieril del mismo.

Cuestionario:

- ¿Le facilita el sistema el trabajo ingenieril con respecto a los productos de trabajo que deben de ser confeccionados durante el procedimiento de pruebas?
- ¿Según su criterio, el sistema modela de forma correcta los productos de trabajo contenidos en la disciplina de Pruebas?
- ¿Si usted fuera a realizar un proyecto utilizaría el sistema propuesto para llevar a cabo el procedimiento de pruebas?
- ¿Considera adecuada la selección de productos de trabajo del Expediente de Proyecto de Desarrollo UCI en su versión 4.0 que genera el sistema? ¿Cree que debería incluirse algún otro, cuál (es)?

Glosario de términos

[1] Proceso de desarrollo definido y por etapas cuyo fin es la terminación de un software.

[2] Rama de la Ingeniería del Software dedicada a estudiar el trabajo con los requisitos en aras de hacer el trabajo con los mismos más fácil y correcto

[3] CRUGE es una extensión para gestión de usuarios y control de acceso basado en roles para el *framework* Yii. Permite administrar y controlar de manera segura los usuarios y los roles que ellos deban tener en una aplicación *web*.