

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS



Universidad de las Ciencias
Informáticas

Facultad 1

**Herramienta para el diseño de casos de prueba funcionales aplicando
técnicas de reutilización**

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

Autor:

Marylaura Martell León

Tutores:

Msc. Alionuska Velázquez Cintra

Ing. Yadelis Velázquez Godoy

La Habana, noviembre de 2021

“Año 63 de la Revolución”

Dedico este trabajo de diploma a mis padres, por ser mi ejemplo de superación, mi apoyo, por luchar para que yo sea quien soy hoy, por demostrarme que los sueños se alcanzan con esfuerzo.

Agradezco a mis padres y mi hermano por ayudarme a ser mejor, por darme la fuerza y la inspiración que necesito.

A mi novio Yohan por acompañarme en todo momento a pesar de los obstáculos para llegar aquí.

A mis amigos: Anaelis, Roxatne y Cristhiam por compartir tantos momentos especiales y por ser la familia que elegí.

A mis compañeros de aula por cada experiencia vivida durante estos 5 años.

A mis tutoras Alionuska y Yadelis por ayudarme en este largo proceso.

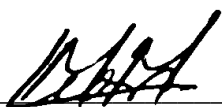
A la FEU por hacerme mejor estudiante y mejor persona y a Chiri por enseñarme todo lo que se de esta maravillosa organización.

A la Revolución, a la UCI y en especial a la Facultad 1 por acogerme y enseñarme que ahí también se puede ser una gran familia.

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo **Marylaura Martell León**, con carné de identidad **98092608979** soy el autor principal del trabajo titulado “Herramienta para la generación de casos de prueba funcionales aplicando técnicas de reutilización” y 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 firmamos la presente a los _____ días del mes de _____ de _____



Marylaura Martell León

Autor

Msc. Alionuska Velázquez Cintra

Tutor

Ing. Yadelis Velázquez Godoy

Tutor

RESUMEN

Las empresas desarrolladoras de software según pasa el tiempo sienten la necesidad de tener producto con mejor calidad, para lograrlo utilizan pruebas de software con el objetivo de identificar posibles errores teniendo en cuenta los requisitos funcionales y no funcionales con los que debe contar el sistema informático deseado por el cliente. En la Universidad de las Ciencias Informáticas se le realiza un control de calidad a todos los productos que se desarrollan antes de ser entregados a los clientes, para ello se necesita hacer pruebas funcionales, es de vital importancia realizar un buen diseño de estas pruebas. Actualmente se emplea mucho esfuerzo en el diseño de los casos de pruebas ya que tienen que hacerlos de forma manual para cada producto, si este proceso se lleva a cabo de manera incorrecta puede influir mucho en la calidad final. En el presente trabajo de diploma se desarrolla una herramienta para el diseño de casos de pruebas funcionales utilizando técnicas de reutilización. La herramienta permite crear y reutilizar casos de prueba, escenarios, variables, disminuyendo así el esfuerzo empleado en este proceso. Se muestran resultados de las distintas etapas del proceso de desarrollo, dígame levantamiento de requisitos, análisis, diseño e implementación de la propuesta. Se describen las pruebas que se realizan con el objetivo de validar la propuesta de solución.

Palabras claves: casos de pruebas, reutilización, pruebas de software

ÍNDICE

INTRODUCCIÓN	1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA	6
1.1 La reutilización de conocimientos en el desarrollo de software	6
1.1.1 Enfoques de la reutilización	6
1.1.2 La reutilización como proceso	7
1.1.3 Técnicas de reutilización	8
1.1.4 Técnicas de diseño de bases de datos y reutilización	10
1.2 Las pruebas funcionales de software: evolución y conceptos	11
1.2.1 Diseño de pruebas de software automatizadas	12
1.2.2 Herramientas para la generación de casos de pruebas funcionales: uso de técnicas de reutilización	13
1.3 Diseño de pruebas de software y reutilización de elementos	15
1.4 Desarrollo de pruebas de software funcionales en la UCI	16
1.5 Metodologías y herramientas para el desarrollo de la solución	17
1.5.1 Metodologías de desarrollo de software	17
1.5.2 Modelado de software	19
1.5.3 Herramienta para el diseño de prototipos	20
1.5.4 Lenguaje de programación	20
1.5.5 Framework	22
1.5.6 Sistema gestor de base de datos	23
1.5.7 Entorno Integrado de desarrollo	23
1.5.8 Servidor web	24
1.6 Conclusiones del capítulo	24
CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN	25
2.1 Descripción de la herramienta	25
2.2 Definición de requisitos de la herramienta	26
2.2.1 Levantamiento de los requisitos	27
2.2.2 Especificación de requisitos funcionales	27
2.2.3 Especificación de requisitos no funcionales	33
2.2.4 Descripción de requisitos mediante HU	35
2.3 Diseño de la propuesta de solución	39
2.3.1 Arquitectura de software	39
2.3.2 Patrones de diseño	40

2.3.3 Estándares de codificación	41
2.3.4 Modelo de datos.....	42
2.4 Conclusiones del capítulo.....	43
CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DEL SISTEMA.....	45
3.1 Implementación de la propuesta de solución	45
3.2 Diagrama de despliegue.....	45
3.3 Pruebas de software.....	46
3.3.1 Estrategia de pruebas	46
3.3.2 Diseño de los escenarios para las pruebas	48
3.4 Desarrollo de pruebas de software	49
3.4.1 Pruebas funcionales.....	49
3.4.3 Pruebas no funcionales.....	51
Pruebas de Usabilidad	51
3.4.4 Triangulación metodológica	52
3.5 Conclusiones del capítulo.....	52
CONCLUSIONES	53
RECOMENDACIONES	54
REFERENCIAS BIBLIOGRÁFICAS.....	55
ANEXOS.....	64
A Cuestionario para obtener información sobre la reutilización de elementos durante el DCP funcionales.....	64
B Guía de entrevista para obtener información sobre la reutilización de elementos durante el DPC funcionales como actividad del proceso de pruebas	66
C Principios básicos que guían las pruebas del software.....	67
D Requisitos correspondientes a cada iteración.....	68
E Diccionario de Datos.....	70
F Corrección luego de ejecutadas las pruebas de regresión	72

ÍNDICE DE TABLAS

Tabla 1. 1:Lista de técnicas de reutilización (Sommerville, 2005).....	8
Tabla 1. 2: Herramientas para la generación de casos de pruebas [Elaboración propia].	13
Tabla 2. 1: Descripción de requisitos funcionales [Elaboración propia].	28
Tabla 2. 2: Descripción de requisitos no funcionales [Elaboración propia].	33
Tabla 2. 3: Asignación de permisos por roles [Elaboración propia].	34
Tabla 2. 4: Historia de usuario #1 [Elaboración propia]	36
Tabla 2. 5: Historia de usuario #2 [Elaboración propia]	37
Tabla 2. 6: Historia de usuario #3 [Elaboración propia]	37
Tabla 2. 7: Estimación de dos puntos por HU [Elaboración propia]	39
Tabla 3. 1: Tiempo de implementación de las HU de ambas iteraciones [Elaboración propia].	.45
Tabla 3. 2: Tipos de prueba del sistema (DCS).....	47
Tabla 3. 3: Cronograma de planificación de pruebas [Elaboración propia].	48
Tabla 3. 4: Relación de Casos de Pruebas con HU [Elaboración propia]	48
Tabla 3. 5: Resultado de la lista de chequeo [Elaboración propia].	51
Tabla 3. 6: Esfuerzo dedicado al diseño de escenarios [Elaboración propia]	52

ÍNDICE DE FIGURAS

Figura 1. 1: Fases del proceso de reutilización (Ortega Loaiza, 2017)	7
Figura 1. 2: Actividades del proceso de pruebas (ISTQB, 2012).	12
Figura 1. 3: Pasos para la automatización de pruebas. [Elaboración propia]	13
Figura 1. 4: Fases e iteraciones de la metodología AUP-UCI (Sánchez, 2015).....	18
Figura 1. 5: Representación gráfica de los escenarios de la metodología AUP-UCI: [Elaboración propia].....	19
Figura 2. 1: Diseño de casos de pruebas funcionales [Elaboración propia].....	25
Figura 2. 2: Diseño de casos de pruebas funcionales utilizando la herramienta [Elaboración propia].....	26
Figura 2. 3: Estimación por dos puntos (Müller y Friedenber, 2011).....	39
Figura 2. 4: Funcionamiento de la arquitectura MVC (Ramírez, 2017).	40
Figura 2. 5: Modelo de datos [Elaboración propia].	43
Figura 3. 1: Diagrama de despliegue [Elaboración propia].	46
Figura 3. 2: Diseño del CP Autenticar usuario [Elaboración propia]	49

ÍNDICE DE CÓDIGO FUENTE

Código fuente 2. 1: Definición de clases	42
Código fuente 2. 2: Definición de métodos.....	42
Código fuente 2. 3: Declaración de variable sin inicializar.....	42
Código fuente 2. 4: Declaración de variable inicializada.....	42

INTRODUCCIÓN

En el mundo global de hoy, donde los mercados están completamente internacionalizados, el entorno cambia rápidamente y la competencia es más fuerte que nunca; las empresas luchan por obtener y mantener una ventaja competitiva. Dado que Internet y los medios de comunicación han permitido que la información sea global y que se pueda obtener sin esfuerzo, la mayoría de las empresas tienen acceso a los mismos procesos y sistemas de gestión, convirtiéndose en el capital intelectual el que queda para marcar la diferencia entre el desempeño de las empresas (Montesa Rausell, 2017).

La competitividad del mercado de la industria del software exige a las organizaciones elaborar productos en menos tiempo de desarrollo, con bajos costos y una alta calidad. Aplicar buenas prácticas de gestión de reutilización de software en el desarrollo del producto trae consigo ventajas competitivas en indicadores de rendimiento de los procesos como son la disminución del tiempo de desarrollo y aumentar la calidad en el producto final (Oro, Alvarado y Ramírez Pérez, 2019).

Uno de los desafíos centrales en el desarrollo de software, justamente es la reutilización, lo que permite utilizar nuevamente uno o más artefactos realizados como parte del desarrollo de un nuevo producto o sistema (Cambarieri, Vivas y García Martínez, 2019).

La reutilización, es una característica promovida por la ingeniería de software que permite el desarrollo de productos de software lo suficientemente flexibles y que pueden ser utilizados con posterioridad en otros desarrollos, ampliando y mejorando sus funcionalidades (Gaitán Peña, 2017).

El objetivo de la reutilización de software tradicional es producir una pieza de software con suficiente flexibilidad para ser utilizada al menos dos veces (Flood, 2017). La reutilización es una alternativa para desarrollar software de forma más eficiente, productiva y rápida (Ortega Loaiza, 2017).

La reutilización abarata los costos del desarrollo: en primer lugar, porque no se necesita implementar una solución de la que ya se dispone; en segundo lugar, porque aumenta la productividad, al poder dedicar los recursos a otras actividades más en línea con el negocio; en tercer lugar, porque probablemente el elemento que reutilizamos ha sido suficientemente probado por su desarrollador, con lo que los *testers* podrán dedicarse a probar otras partes más críticas del sistema, obteniendo entonces unos niveles de calidad mucho más altos que si el sistema se desarrolla desde cero (Polo Usaola, 2013).

El desarrollo tecnológico actual y la percepción de calidad adquirida por los usuarios finales y clientes, nos advierte una revolución en los procesos de pruebas de software. El diseño de las pruebas adquiere mayor relevancia a partir del creciente uso de las tecnologías emergentes, pues señalan la automatización como elemento importante para lograr el cumplimiento de los objetivos del proceso de pruebas y reorientar esfuerzos hacia áreas de mayor prioridad en función de la calidad deseada en el producto final (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020). El diseño de pruebas incluye la identificación de los escenarios o casos de pruebas que serán probados.

Los Casos de Prueba (CP) son un conjunto de pasos (condiciones y/o variables) y/o resultados esperados con las cuales un analista probará y definirá si un software es parcial o completamente satisfactorio (Anaya, 2020). Pueden ser diseñados utilizando un enfoque dinámico o un enfoque estático. Dentro del enfoque dinámico existen diferentes métodos. Las pruebas de caja blanca (*white box*) derivan pruebas basadas en la estructura o implementación interna del sistema. Las pruebas de caja negra (*black box*) derivan condiciones de prueba y casos de prueba para la funcionalidad del componente o sistema (ISTQB, 2018).

El desarrollo científico y tecnológico de un país puede definirse como el proceso de acumulación de capacidades y actitudes de la sociedad para generar, asimilar, adaptar, perfeccionar, apropiar y aplicar conocimientos y sus correspondientes tecnologías. En este contexto y el de los documentos resultantes del VII Congreso del Partido, refrendados por el Parlamento cubano en el 2017, se destaca la importancia de la aplicación de la ciencia y la innovación al sector productivo y social (Santos Alonso, 2020).

El avance de la ciencia y la tecnología aplicadas en favor de la sociedad contribuye al desarrollo sostenible de un país. Uno de los elementos fundamentales para trabajar en función de la soberanía tecnológica en Cuba es contar con sistemas desarrollados por especialistas nacionales. La soberanía tecnológica es el derecho y el deber de una nación de dominar sus medios tecnológicos a tal punto que no puedan ser controlados de manera injerencista por otros intereses ajenos al bienestar de su desarrollo (Albo Castro y Coca Bergolla, 2020).

La Universidad de las Ciencias Informáticas (UCI) como parte del proceso de informatización de la sociedad cubana, cuenta con una red donde se integran 10 centros de desarrollo de software, el Centro de Soporte y la Dirección de Calidad. El trabajo en estrecha interrelación ha permitido el desarrollo de productos que impactan en el proceso de Informatización de la Sociedad Cubana. Además, cuenta con un Laboratorio de Pruebas de Software (LPS) el cual se centra en la ejecución de pruebas de sistema donde verifica la calidad del producto final de acuerdo a lo

establecido por la familia de normas ISO/IEC 25000 y en correspondencia con los requerimientos del sistema, establecidos de mutuo acuerdo con el cliente (Velázquez Cintra, Feble Estrada y Merced Len, 2018; Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).

Se desarrolló una herramienta que permitió a los usuarios diseñar casos de pruebas disminuyendo así el esfuerzo respecto al tiempo y recursos humanos empleados en el desarrollo de esta actividad (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020). La herramienta permite particionar las variables y reutilizarlas en la construcción de los escenarios.

Se aplicó una encuesta a 8 especialistas de la UCI (Anexo A Cuestionario para obtener información sobre la reutilización de elementos durante el DCP funcionales.) con el objetivo de indagar sobre la importancia de la etapa de diseño de CP, el esfuerzo que se emplea en la misma, además identificar los elementos que pueden ser reutilizados durante este proceso. Los encuestados tenían más de 7 años de experiencia en actividades relacionadas con el proceso de desarrollo de software en la universidad y han participado en más de 10 proyectos vinculados a las tareas de diseño de CP. En la aplicación de este instrumento se contó con la participación de 5 centros de desarrollo. Se arribó a las siguientes conclusiones:

- El 100 % de los encuestados consideran que el diseño incorrecto de los casos de prueba influye en la calidad del producto.
- El 100 % de los encuestados han identificado elementos que se pueden reutilizar.
- El 87,5 % de los encuestados le otorga mucha importancia a la reutilización en el proceso de diseño de casos de prueba.
- El 75 % de los encuestados concuerdan con que emplean mucho esfuerzo en el diseño de casos de pruebas funcionales.
- El 100 % de los encuestados consideran oportuno contar con una herramienta que le permita crear plantillas genéricas para facilitar el diseño de casos de pruebas.
- El 87,5 % de los encuestados no utilizan herramientas para generar casos de prueba.

Los análisis realizados demuestran que, en contextos como la UCI, donde los centros de desarrollo especializan sus procesos en función de tipos de aplicaciones, esa demora de tener que realizar todos los procesos de forma manual provoca atrasos en los cronogramas de trabajo de los proyectos, resulta recomendable definir el uso de plantillas que permitan la reutilización de conceptos más amplios: escenarios, casos de pruebas y grupos de Diseño de Casos de Pruebas(DCP) por tipos de productos; con el objetivo de disminuir el esfuerzo empleado por los analistas para generar todos los casos de prueba de un determinado producto.

Teniendo en cuenta la situación descrita anteriormente, se plantea como **problema de la investigación**: ¿Cómo automatizar el proceso de diseño de casos de pruebas funcionales en la UCI?

Del planteamiento anterior se deriva como **objeto de estudio** de esta investigación: el diseño de casos de pruebas funcionales, enfocando el **campo de acción** en: las técnicas de reutilización durante el diseño de casos de pruebas funcionales.

Teniendo en cuenta el problema a resolver se define como **objetivo general**: Desarrollar una aplicación web para el diseño de casos de prueba funcionales en la UCI aplicando técnicas de reutilización.

De este se derivan los siguientes **objetivos específicos**:

1. Elaborar el marco teórico-referencial sobre herramientas para el diseño de casos de prueba funcionales.
2. Diseño e implementación de la herramienta para el diseño de casos de prueba funcionales.
3. Validar la herramienta propuesta.

Par dar cumplimiento a los objetivos específicos planteados se trazaron las siguientes **tareas de investigación**:

1. Estudio de las técnicas de diseño de casos de casos de pruebas funcionales.
2. Estudio de las técnicas para la reutilización de conceptos en el desarrollo de software.
3. Valoración de herramientas de DCP.
4. Elaboración de productos de trabajo como resultado de las actividades de análisis y diseño de casos de prueba.
5. Implementación de la propuesta de solución.
6. Validación de la herramienta a partir de la ejecución de pruebas.

Los **métodos investigativos** empleados para la realización de esas tareas son:

Métodos teóricos:

- Histórico-lógico: para analizar los conceptos asociados al diseño de casos de pruebas funcionales y la reutilización de elementos durante este proceso.
- Analítico-sintético: se utiliza para el estudio de trabajos similares e investigaciones que abordan el tema de la reutilización de elementos y el diseño de casos de pruebas utilizándolos como punto de partida para el desarrollo de la solución.

Métodos empíricos:

- Encuestas: para obtener información sobre la reutilización de elementos durante el DCP funcionales. Identificar los elementos que requieren ser reutilizados durante el diseño de casos de prueba. El instrumento utilizado fue el cuestionario. [Anexo A]
- Entrevista: para determinar qué elementos pueden ser reutilizados durante el diseño de casos de pruebas funcionales y evaluar la importancia que se le concede a la aplicación de técnicas de reutilización en este proceso. El instrumento utilizado fue la guía de entrevista. [Anexo B]

Como método cualitativo-cuantitativo se empleó el análisis porcentual para evaluar los resultados de las encuestas y entrevistas realizadas.

El presente trabajo de diploma se estructura en introducción, tres capítulos, conclusiones y recomendaciones. Además, se agregan los anexos que, a consideración de la autora, permiten comprender mejor la investigación realizada.

En el *Capítulo 1. Fundamentación teórica*, se realiza un análisis del concepto de reutilización de en el desarrollo de software, se estudian las técnicas de diseño de casos de pruebas funcionales mediante el uso de técnicas para la reutilización desde la base de datos, además se realiza un estudio de las herramientas homólogas escogiendo la más adecuada para darle cumplimiento a la propuesta de solución.

El *Capítulo 2. Análisis y diseño de la propuesta de solución*, describe el desarrollo de las actividades correspondientes al análisis y diseño de la herramienta. Se explican las características principales de los productos de trabajo generados como resultado de esas actividades.

El *Capítulo 3. Implementación y pruebas del sistema*, se muestra una descripción de la validación de la propuesta. Se describe la estrategia de pruebas utilizada y los resultados obtenidos de las mismas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se fundamenta toda la base teórica de la presente investigación. Se realiza un análisis sobre el desarrollo de las pruebas funcionales, el diseño de pruebas, la reutilización de elementos en el diseño de estas pruebas. Además, se estudian las condiciones para el desarrollo de estas pruebas en el entorno UCI. Se analizan las herramientas que resuelven problemáticas similares con el objetivo de evaluar su adaptabilidad a este contexto. Se plantea la metodología utilizada y se realiza una fundamentación de las tecnologías, herramientas y lenguajes a utilizar en todo el proceso de desarrollo de la solución informática.

1.1 La reutilización de conocimientos en el desarrollo de software

La reutilización aparece a finales de la década del 60 como una alternativa para superar la crisis del software (Anaya de Páez, 2012). Desde sus orígenes, la reutilización ha sido valorada como una vía para superar la crisis del software y se ha propugnado como mecanismo para reducir el tiempo de desarrollo, incrementar la productividad de los desarrolladores y reducir la densidad de errores (Delgado González, 2016).

La reutilización de componentes software permite agilizar los procesos de producción y despliegue de las aplicaciones web. Dichos componentes han sido debidamente probados y verificados con anterioridad, ayudando a crear sistemas informáticos más confiables y seguros (Vargas Fandiño, Sandoval Ramírez y Vera Rivera, 2020).

Todo ingeniero de software, debe ser capaz de identificar estas situaciones que se presentan frecuentemente y que han sido ya, con toda probabilidad, resueltas con antelación por otras personas de manera eficiente. Este tipo de conocimiento, en el que se describen problemas que aparecen de manera más o menos frecuente, y que incluyen en su descripción una o más buenas maneras de resolverlo, conforma lo que se llama un patrón (Polo Usaola, 2013).

La reutilización es de gran utilidad ya que brinda la posibilidad de crear proyectos de forma más rápida, con una alta calidad y empleando un tiempo menor, mejora la productividad del equipo.

1.1.1 Enfoques de la reutilización

Existen dos enfoques bien distintos en cuanto a reutilización: el enfoque oportunista en el que se utilizan partes que se ajustan al problema en cuestión, obtenidos de un repositorio en el que fueron almacenados estos componentes que esperan ser usados; en un enfoque más planificado o proactivo, implica una inversión inicial para obtener beneficios a futuro, creando componentes reutilizables y genéricos para ser utilizados fácilmente, es decir, hay una planificación (Montero Cáceres, 2017).

La reutilización oportunista es de interés cuando se necesita construir sistemas que sean grandes, complejos, fiables, menos costosos y que se terminen a tiempo. Es una forma de incrementar la productividad en el desarrollo de software y su calidad (Ordoñez Guzmán y Giraldo Muñoz, 2016). La reutilización sistemática de software puede definirse como el desarrollo de software a partir de una colección de componentes básicos que aprovechan las similitudes en los requisitos, la arquitectura o el diseño (Flood, 2017).

Después de analizar los dos enfoques de la reutilización de software se concluye que es recomendable emplear los dos, teniendo en cuenta la situación en la que se encuentre el equipo de desarrollo puede tomarse el tiempo de hacer una planificación detallada o reutilizar componentes que ya se encuentren desarrollados.

1.1.2 La reutilización como proceso

Durante la automatización, encontraremos en ocasiones algunas pruebas que realizan operaciones muy similares, e incluso cuenten con algunas fases de la ejecución idénticas. Se buscará siempre que sea posible generalizar el comportamiento creando pruebas base. Estas clases base implementan la funcionalidad común a todas las pruebas que hereden de ellas, evitando así repetirla (Vega Llobell, 2018).

El proceso de reutilización tiene varias fases de suma importancia (Ortega Loaiza, 2017):

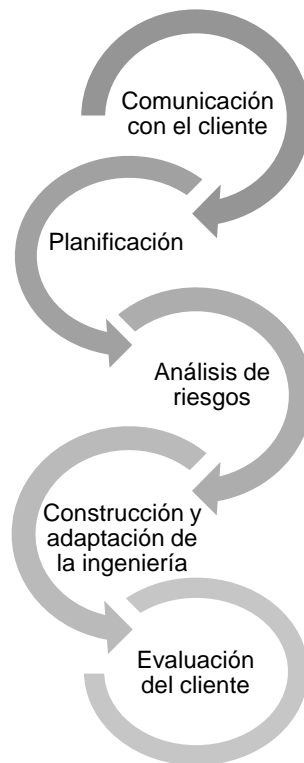


Figura 1. 1: Fases del proceso de reutilización (Ortega Loaiza, 2017)

Antes de comenzar a desarrollar lo que se quiere reutilizar tiene que pasar por un proceso de planificación para evitar fallas inesperadas. Para ello hay que tener en cuenta una serie de factores que se describen a continuación (Sommerville, 2005):

- La agenda de desarrollo del software: Si el software tiene que desarrollarse rápidamente, debería intentarse reutilizar sistemas comerciales en vez de componentes individuales.
- Vida esperada del software: Si se está desarrollando un sistema de larga vida, habría que centrarse en la mantenibilidad del sistema.
- Los conocimientos, habilidades y experiencia del grupo de desarrollo: Todas las tecnologías de reutilización son bastante complejas y se necesita bastante tiempo para comprenderlas y usarlas de forma efectiva. Sin embargo, si el grupo de desarrollo posee habilidades en un área particular, probablemente habría que centrarse en ella.
- La criticidad del software y sus requerimientos no funcionales: Para un sistema crítico que tiene que ser certificado por un regulador externo, se tiene que crear un caso de confiabilidad para el sistema
- El dominio de las aplicaciones: En algunos dominios de aplicaciones como los sistemas de información médica y de fabricación, hay varios productos genéricos que pueden reutilizarse para configurarlos a una situación particular.
- La plataforma sobre la que el sistema se va a ejecutar: Algunos modelos de componentes, como COM/Active X, son plataformas específicas de Microsoft. Si se está desarrollando sobre una plataforma como éstas, esta aproximación puede ser la más adecuada.

Las fases de la reutilización están estrechamente relacionadas con las fases del proceso de desarrollo de software por lo que es recomendable alinearlas. En este caso lo primero que se hace es comunicarse con el cliente para obtener los requisitos con los que debe cumplir el producto, se debe realizar una buena planificación para evitar cometer errores que agraven los costos del proyecto, se analizan todos los riesgos que se enfrentarán durante la implementación, se desarrolla el producto y por último después de pasar las pruebas, el cliente debe evaluar el producto.

1.1.3 Técnicas de reutilización

Las técnicas para la reutilización son muy diversas, para decidir cuál se desea emplear en un sistema hay que tener en cuenta los requerimientos del sistema a desarrollar, la tecnología, activos reutilizables disponibles y la experiencia del grupo de desarrollo (Sommerville, 2005).

Tabla 1. 1:Lista de técnicas de reutilización (Sommerville, 2005)

Técnicas	Descripción
----------	-------------

Patrones arquitectónicos	Arquitecturas estándar de software que soportan tipos comunes de sistemas de aplicaciones son utilizadas como la base de aplicaciones.
Patrones de diseño	Abstracciones genéricas que ocurren a través de aplicaciones que son representadas como patrones de diseño mostrando objetos e interacciones abstractas y concretas
Desarrollo basado en componentes	Los sistemas son desarrollados integrando componentes (colección de objetos) que se ajustan a los estándares de componentes en modelos
Estructuras de aplicación	Colección de clases abstractas y concretas son adaptadas y extendidas para crear sistemas de aplicación
Envoltura de sistemas heredados	Los sistemas heredados son envolturas o cubiertas definiendo un grupo de interfaces y permitiendo el acceso a éstos sistemas heredados a través de éstas interfaces.
Sistemas orientados al servicio	Los sistemas son desarrollados mediante vinculación de los servicios compartidos, que pueden ser proporcionados externamente.
Líneas de producto de software	Un tipo de aplicación es generalizado alrededor de arquitecturas comunes por lo que pueden ser adaptadas para diferentes clientes.
COTS reutilización de productos	Los sistemas son desarrollados mediante la configuración e integración existente en sistemas de aplicaciones.
Sistemas ERP	Sistemas de gran escala que encapsulan funcionalidad, y reglas genéricas empresariales son configuradas por una organización.
Aplicaciones verticales configurables	Sistemas genéricos son diseñados de tal forma que puedan ser configurados a las necesidades específicas del sistema del cliente.
Biblioteca de programas	Biblioteca de clases y funciones ,que comúnmente se emplean utilizando abstracciones , están disponibles para reutilizar.
Ingeniería dirigida por modelos	El software se representa como modelos de dominio y modelos independientes de aplicación el código se genera a partir de estos modelos.

Generadores de programas	Un sistema generador contiene conocimiento de un tipo de aplicación y es utilizado para generar sistemas en ese dominio donde un usuario suplente un modelo de sistema.
Desarrollo de software orientado a aspectos	Los componentes compartidos se enlazan en una aplicación en diferentes lugares cuando se compila el programa.

Es recomendable el uso de una o varias técnicas de reutilización dependiendo del contexto donde se vaya a emplear.

Varios autores (Gaitán Peña, 2017), (Vázquez-Ingelmo y Therón, 2020) señalan a las líneas de producto de software como la técnica más utilizada ya que permiten el desarrollo de diversos modelos o sistemas software los cuales comparten entre sí características a partir de un núcleo común altamente reutilizable, permite identificar, a través de ingeniería de dominio y la abstracción de este, puntos comunes entre los sistemas que conforman el espacio de posibles productos. Considerando las particularidades de la presente investigación es posible señalar esa técnica como la más adecuada.

1.1.4 Técnicas de diseño de bases de datos y reutilización

Si hablamos de una base de datos en el contexto informático, hay que señalar que se trata de un programa o archivo electrónico en el que la información va organizada y estructurada en determinados campos que serán de utilidad para el usuario (Peiró, 2020).

La normalización es el proceso de organización de datos en una base de datos. Esto incluye crear tablas y establecer relaciones entre dichas tablas de acuerdo con las reglas diseñadas tanto para proteger los datos como para que la base de datos sea más flexible al eliminar la redundancia y la dependencia incoherente. Existen tres reglas para lograr dicha normalización (MaryQiu1987, 2021):

- Primer formulario normal:
 - Eliminar los grupos de repetición en tablas individuales.
 - Crear una tabla independiente para cada conjunto de datos relacionados.
 - Identificar cada conjunto de datos relacionados con una clave principal.
- Segundo formulario normal:
 - Crear tablas independientes para conjuntos de valores que se aplican a varios registros.
 - Relacionar esas tablas con una clave externa.
- Tercer formulario normal:
 - Elimine los campos que no dependen de la clave.

La normalización de base de datos es un punto muy importante que deberíamos de tomar muy en serio para establecer cimientos sólidos sobre los cuales podemos construir aplicaciones robustas que en el futuro no presenten problemas de base de datos difíciles de solucionar (Sarmiento, 2017).

Tener una base de datos normalizada brinda la posibilidad de poder reutilizar los elementos almacenados en ella.

Tener una base de datos lista para ser reutilizada es una ventaja ya que brinda la posibilidad de acceder a los datos y reutilizarlos en varios proyectos, teniendo en cuenta los elementos y relaciones en común.

1.2 Las pruebas funcionales de software: evolución y conceptos

Las Pruebas o *Testing* de Software se trata básicamente del conjunto de actividades dentro del desarrollo de un software permitiendo así tener procesos, métodos de trabajo y herramientas para identificar oportunamente los defectos en el software, logrando la estabilidad del mismo. Siendo el único instrumento capaz de precisar la calidad de un producto de software, es decir, es el único procedimiento con el que se puede garantizar que un software cumple con los requerimientos solicitados por los usuarios (Juárez, 2020).

Existen enfoques dinámicos y estáticos para las pruebas. Los enfoques dinámicos apuntan a ejecutar una parte o todo el software para determinar si funciona según lo esperado. El enfoque estático se refiere a la evaluación del software sin ejecutarlo usando mecanismos automatizados (herramientas asistidas) y manuales tales como controles de escritorio, inspecciones y revisiones (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).

Las pruebas de software son una parte integral del ciclo de vida del desarrollo de software (SDLC). Son la forma en que puede estar seguro acerca de la funcionalidad, el rendimiento y la experiencia del usuario (Lee, 2020).

Dichas pruebas son un elemento crítico para la garantía del correcto funcionamiento del software. Entre sus objetivos están (ISTQB, 2018):

1. Detectar defectos en el software.
2. Verificar la integración adecuada de los componentes.
3. Verificar que todos los requisitos se han implementado correctamente.
4. Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
5. Diseñar CP que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

Para lograr los objetivos propuestos, un ingeniero de software deberá conocer los principios básicos que guían las pruebas del software (Müller y Friedenber, 2011), los cuales pueden ser consultados en el Anexo C (ISTQB, 2018).

Se puede describir de forma general cómo funciona este proceso, a partir de la relación de las actividades que lo componen. En la figura 2 pueden observarse estas actividades:



Figura 1. 2: Actividades del proceso de pruebas (ISTQB, 2012).

Para los efectos de esta investigación las pruebas serán consideradas como la ejecución lógica de actividades en las que un sistema o componente es ejecutado de forma dinámica bajo condiciones específicas, previamente definidas en DCP, se observan y almacenan los resultados obtenidos para ser comparados con los esperados y emitir una evaluación de algún aspecto del sistema o componente.

Con la llegada del desarrollo de aplicaciones emergentes se impone la búsqueda de nuevas formas de pruebas que permitan verificar los elementos básicos de las aplicaciones empleando el menor esfuerzo posible. En este caso la automatización de las pruebas alcanza mayor relevancia.

1.2.1 Diseño de pruebas de software automatizadas

La automatización de pruebas de software es una de las opciones más fascinantes para enfrentar un equilibrio habitual de entregas al cliente sin comprometer la calidad del producto software; esto se hace más necesario cuando ya se tiene un producto en un ambiente de producción y con una gran cantidad de usuarios que pueden verse afectados por un fallo (Rueda Patiño, Cruz Mosquera y Londoño Rojas, 2016).

A medida que los proyectos de software se vuelven cada vez más complejos y entrelazados y con la gran cantidad de plataformas y dispositivos diferentes que necesitan ser probados, es más importante que nunca tener un proceso robusto para administrar sus actividades de prueba y asegurarse de que se estén utilizando recursos de prueba limitados enfocado en las áreas de

mayor riesgo e importancia. El software de gestión de pruebas le ayuda a gestionar ese proceso (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).

Existen varios pasos que se deben tener en cuenta a la hora de diseñar una prueba automatizada (Colorado Rivera, 2020):

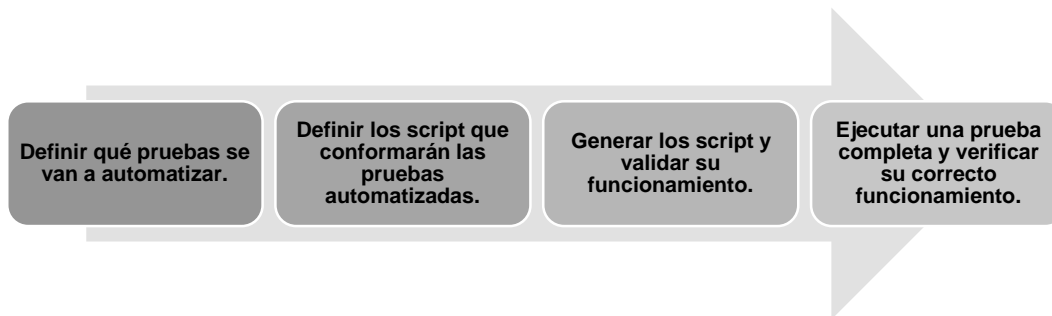


Figura 1. 3: Pasos para la automatización de pruebas. [Elaboración propia]

El diseño de pruebas automatizadas ha sido una alternativa para las pruebas manuales, ya que estas consumen demasiado tiempo del proceso. Automatizar estas pruebas puede mejorar considerablemente la calidad del software, brindando la posibilidad de emplear más tiempo en detalles más importantes. Reducen los errores humanos durante la etapa de pruebas en el ciclo de desarrollo.

1.2.2 Herramientas para la generación de casos de pruebas funcionales: uso de técnicas de reutilización

En la actualidad los proyectos a desarrollar pueden llegar a ser tan grandes que es necesario utilizar herramientas con las que se puedan administrar todos los casos de prueba, crearlos, gestionarlos, incluso a veces se requiere exportar los resultados de las pruebas que se le aplican al software. Existen muchas herramientas que pueden ayudar a agilizar este proceso.

En la tabla a continuación se muestran algunas herramientas para la creación de casos de pruebas:

Tabla 1. 2: Herramientas para la generación de casos de pruebas [Elaboración propia].

Nombre de la herramienta	Open Source	Principales funcionalidades	Reutilización
Selenium (Coelho, 2019)	Sí	<ul style="list-style-type: none"> Tiene un conjunto de herramientas de software y cada una muestra una perspectiva diferente. 	Con las pruebas de regresión se consiguen pruebas automatizadas

		<ul style="list-style-type: none"> • Permite grabar, editar y depurar casos de pruebas que se pueden automatizar. • Brinda almacenamiento en varios formatos los <i>test</i> realizados. 	que luego se pueden reutilizar.
TestLink (Leave a Comment, 2019)	Sí	<ul style="list-style-type: none"> • Permite crear y gestionar casos de prueba, organizarlos en planes de pruebas. • Permite realizar un seguimiento de los resultados, establecer trazabilidad con los requisitos, generar informes. 	Emplea la reutilización a través de Test Suits elementos como: nombre de la versión del software.
Rational Quality Manager (IBM Corporation, 2021)	No	<ul style="list-style-type: none"> • Herramienta basada en web que ofrece funciones integrales de planificación de pruebas, construcción de pruebas y administración de artefactos durante todo el ciclo de vida del desarrollo de software. 	Emplea la reutilización mediante el uso de palabras claves.
Redmine (Ramos, 2015)	Sí	<ul style="list-style-type: none"> • Herramienta para la gestión de proyectos y seguimiento de incidencias de código abierto basado en la web. • Permite a sus usuarios gestionar múltiples proyectos y sus respectivos subproyectos. • Proporciona herramientas muy útiles como wikis y foros, seguimiento temporal y control de acceso flexible basado en roles. 	No

<p>Herramienta para el diseño y gestión de casos de pruebas funcionales (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).</p>	<p>Sí</p>	<ul style="list-style-type: none"> • Herramienta para el diseño y gestión de casos de pruebas funcionales en la UCI. • Cuenta con la posibilidad de exportar en formato Excel el DCP realizados a un producto específico. 	<p>Permite la reutilización de variables particionadas en escenarios de casos de prueba.</p>
--	-----------	---	--

Después de realizar el análisis mostrado en la tabla anterior se arriba a las conclusiones siguientes:

- Las herramientas analizadas hacen uso de las técnicas de reutilización principalmente durante la etapa de ejecución de pruebas, al realizar pruebas de regresión.
- Solo la Herramienta para el diseño y gestión de casos de pruebas funcionales aplica la reutilización desde el diseño de los casos de pruebas, sin embargo, reduce su alcance a la reutilización de las variables.

La herramienta más adecuada para utilizar en el entorno donde se desarrolla la investigación es Herramienta para el diseño y gestión de casos de pruebas funcionales, ya que además de cumplir con el criterio de soberanía tecnológica, es la que más se ajusta a las condiciones del entorno donde se realizan las pruebas.

Como parte de la investigación se realizó un estudio detallado de esta herramienta arrojando los siguientes resultados:

- La base de datos no está preparada para aplicar la reutilización de casos de pruebas completos ni escenarios, pero sí permite la escalabilidad de la misma en función de adaptarla al estado deseado.
- No define el uso de plantillas que permitan la reutilización de escenarios, casos de pruebas y grupos de diseño de casos de pruebas (DCP) por tipos de productos.

1.3 Diseño de pruebas de software y reutilización de elementos

El diseño de la prueba es la actividad que define cómo algo debe ser probado. Implica la identificación de CP mediante la elaboración gradual de las condiciones de prueba identificadas o la base de prueba utilizando técnicas de prueba identificadas en la estrategia y/o el plan de prueba. Para un nivel de prueba determinado puede llevarse a cabo una vez que se hayan

identificado las condiciones de la prueba y se disponga de información suficiente para permitir la producción de CP de bajo o alto nivel, según el enfoque empleado para el diseño de la prueba (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).

El correcto diseño de las pruebas es fundamental, de un buen diseño depende que el software tenga la mejor calidad. Al emplear la reutilización en este proceso se disminuye la cantidad de componentes, variables y demás elementos repetidos dando la oportunidad de utilizar aquellos que sean común y ya estén desarrollados.

El método de caja negra es una técnica que prueba la funcionalidad de una aplicación sin tener en cuenta o conocer su implementación, diseño o estructura internas, de manera más cercana a la perspectiva de un usuario del sistema. Así, el foco principal está en observar las salidas que genera el sistema ante unas entradas determinadas para determinar si son correctas, independientemente del estado interno o los resultados intermedios que genere el sistema (Parra Valverde, 2020).

Este método de caja negra es de las actividades más importantes de las pruebas. La prueba de caja negra se centra principalmente en los requisitos funcionales del sistema, por lo que el resultado de las pruebas depende de la calidad de la especificación de los requisitos funcionales. Por la complejidad y la importancia que tiene esta actividad de prueba es recomendable automatizar las mismas. Con el objetivo de lograr un mayor control y disminuir la posibilidad de cometer errores a la hora del diseño, ya que de este depende la calidad de los resultados y del producto en general.

1.4 Desarrollo de pruebas de software funcionales en la UCI

Las pruebas de software que se desarrollan en el LPS de la UCI son guiadas por especialistas que funcionan como Coordinadores bajo un modelo de pruebas que combina las políticas de la actividad desarrollo-producción de la UCI con las características de calidad del producto definidas en la NC ISO/IEC 25010:2016 y las buenas prácticas de la industria (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).

El LPS cuenta con algunas fortalezas dentro de las cuales se encuentran: actividad de desarrollo-producción de software evaluada por el nivel 2 de CMMI-DEV, ejecución de pruebas especializadas, estructura creada a todos los niveles y mejora continua de los procedimientos utilizados para el desarrollo de pruebas especializadas (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).

Durante el proceso de pruebas en la UCI se diseñan una gran cantidad de CP en un corto período de tiempo, por la necesidad de realizar las pruebas de funcionalidad al producto. Se toman los

elementos que son comunes en varios casos de prueba y se vuelven a copiar en el próximo diseño.

Se realizó una entrevista a 8 especialistas de la UCI (Anexo B Guía de entrevista para obtener información sobre la reutilización de elementos durante el DPC funcionales como actividad del proceso de pruebas), con el objetivo de determinar qué elementos pueden ser reutilizados durante el diseño de casos de pruebas funcionales y evaluar la importancia que se le concede a la aplicación de técnicas de reutilización en este proceso. Los entrevistados tenían entre 10 y 14 años de experiencia en actividades relacionadas con el proceso de desarrollo de software en la universidad. Se arribó a las siguientes conclusiones:

- El 100 % de los entrevistados considera necesario reutilizar elementos en el proceso de diseño de casos de pruebas funcionales.
- El 100 % de los entrevistados considera que reutilizando elementos disminuiría el tiempo empleado en diseño de casos de prueba.
- El 87,5 % de los entrevistados reutilizan elementos en el diseño de casos de prueba de forma manual, estos elementos son: precondiciones, campos de entrada, gestión de usuarios y respuesta esperada.
- El 100 % de los entrevistados consideran oportuno contar con una herramienta que le permita crear plantillas genéricas para facilitar el diseño de casos de pruebas.

Después de analizar los resultados de la entrevista aplicada se concluye que es necesario aplicar técnicas de reutilización en casos de prueba, escenarios, pues de esta manera se reduce el tiempo y esfuerzo en el DCP.

1.5 Metodologías y herramientas para el desarrollo de la solución

A continuación, se muestran las metodologías y herramientas a utilizar durante el desarrollo del software, así como las herramientas para el modelado del software y el modelado de los prototipos. Teniendo en cuenta que la propuesta va encaminada a una mejora de la herramienta “Herramienta para el diseño y gestión de casos de pruebas funcionales” (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020) ya desarrollada se asumen las herramientas y tecnologías utilizadas en esa investigación.

1.5.1 Metodologías de desarrollo de software

Una metodología es el proceso que se suele seguir a la hora de diseñar una solución o un programa específico. las metodologías de desarrollo de software son enfoques de carácter estructurado y estratégico que permiten el desarrollo de programas con base a modelos de sistemas, reglas, sugerencias de diseño y guías (Pérez, 2016).

Existen dos clasificaciones: tradicionales y ágiles. Las tradicionales centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido en la fase inicial del desarrollo del proyecto. Las ágiles se basan en dos aspectos fundamentales, retrasar las decisiones y la planificación adaptativa. Destacan en la adaptabilidad de los procesos de desarrollo. Estas metodologías destacan que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan (Ortiz Herrera, 2018).

En el documento “Programa de Mejora Metodología de desarrollo de software para la Actividad Productiva de la UCI” (Sánchez, 2015) se explica dicha metodología, la cual se trata a continuación:

Para el ciclo de vida de los proyectos de la UCI se definió una variante de la metodología AUP a la cual se le otorgó el nombre de AUP-UCI. La metodología AUP propone cuatro fases (inicio, elaboración, construcción, transición), para la variante de la UCI se decide mantener la fase de inicio y se unen las tres fases restantes en una sola llamada ejecución y se agrega la fase de Cierre.

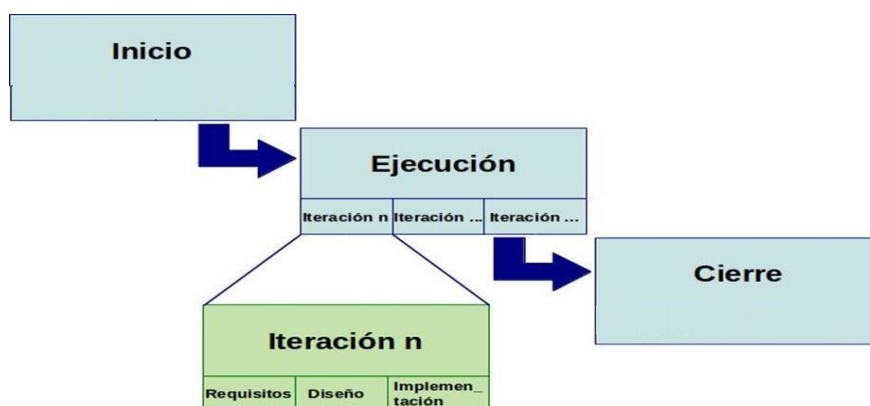


Figura 1. 4: Fases e iteraciones de la metodología AUP-UCI (Sánchez, 2015).

Fases:

Inicio: Se lleva a cabo las actividades relacionadas con la planeación del proyecto. Se realiza un estudio de la organización del cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución: Se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

Cierre: Se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales del cierre del proyecto.

Existen tres formas de encapsular los requerimientos en la variación AUP-UCI: Casos de Uso del Sistema (CUS), Historias de usuario (HU) y Descripción de requerimientos por proceso (DRP), agrupados en cuatro escenarios como se muestra a continuación:

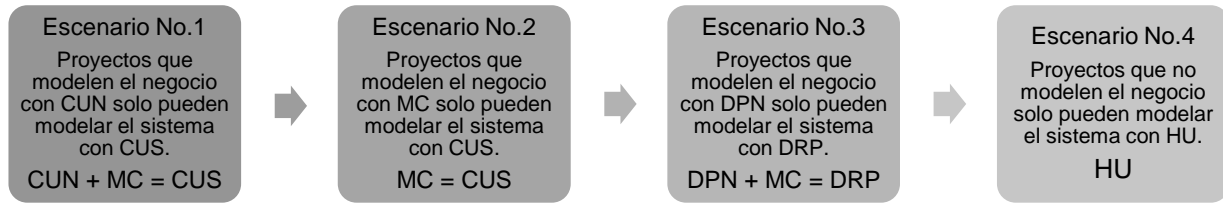


Figura 1. 5: Representación gráfica de los escenarios de la metodología AUP-UCI: [Elaboración propia].

Basado en lo antes expuesto, se decide encapsular los requisitos en el escenario cuatro. Para la selección se tuvo en cuenta que el cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requerimientos y así poder implementarlos, probarlos y validarlos. El tratamiento de las HU es muy dinámico y flexible, lo que permite que estas puedan reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas de una manera más fácil, sin invertir tanto tiempo y esfuerzo.

1.5.2 Modelado de software

La modelación de software es una técnica con la complejidad inherente a los sistemas. El uso de modelos ayuda al equipo de trabajo de desarrollo de software a visualizar el sistema de información a construir. Además, los modelos de un nivel de abstracción mayor pueden utilizarse para la comunicación con el cliente. Un modelo permite comprender mejor el sistema que estamos desarrollando: sus elementos y sus relaciones (Oscoco Huangal y Montoya Maldonado, 2016).

A continuación, se describen las principales características de la herramienta a utilizar para el modelado de la propuesta de solución.

Lenguaje Unificado de Modelado(UML 2.5)

El lenguaje de modelado unificado (UML por sus siglas en inglés) es un estándar para la representación visual de objetos, estados y procesos dentro de un sistema. Por un lado, el lenguaje de modelado puede servir de modelo para un proyecto y garantizar así una arquitectura de información estructurada; por el otro, ayuda a los desarrolladores a presentar la descripción del sistema de una manera que sea comprensible para quienes están fuera del campo (IONOS España S.L.U., 2018).

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*) permiten el modelado de los sistemas mediante diferentes diagramas, generación de código a partir de estos y viceversa.

Además, permiten abstraerse del código fuente, en un nivel donde la arquitectura y el diseño se tornan más fáciles de entender (Velázquez Godoy, Velázquez Cintra y Collado Rolo, 2020).

Después de analizar las distintas herramientas para el modelado, se decide utilizar Visual Paradigm por ser una herramienta que ayuda a crear diagramas con rapidez, se puede utilizar en varios sistemas. Además, la UCI posee licencia para su utilización. A continuación, se brindan detalles de esta herramienta.

Visual Paradigm 15.2

Visual Paradigm es una herramienta de software diseñada para que los equipos de desarrollo de software modelen el sistema de información empresarial y gestionen los procesos de desarrollo. Admite lenguajes y estándares de modelado clave de la industria, como Lenguaje de modelado unificado (UML). Ofrece un conjunto completo de herramientas de software que las empresas necesitan para la captura de requisitos, análisis de procesos, diseño de sistemas, diseño de bases de datos, etc (Visual Paradigm, 2021).

1.5.3 Herramienta para el diseño de prototipos

Los prototipos son una parte integral del proceso de diseño, ya que permiten la revisión de los conceptos básicos del proyecto en la etapa inicial de su desarrollo (Galiana, 2021).

Balsamiq Mockups 3

Es una herramienta gráfica para esbozar interfaces de usuarios, para sitios web, aplicaciones móviles y de escritorio. Está enfocada a la fase de creación, permite diseñar prototipos enfocados en la estructura en lugar de los colores y los íconos (Balsamiq Studios, 2021).

1.5.4 Lenguaje de programación

Un lenguaje de programación nos permite comunicarnos con las computadoras a través de algoritmos e instrucciones escritas en una sintaxis que la computadora entiende e interpreta en lenguaje de máquina. Los lenguajes de programación permiten a las computadoras procesar de forma rápida y eficientemente grandes y complejas cantidades de información (López Mendoza, 2020).

Lenguaje de programación del lado del cliente

Se usan para su integración en páginas web. Un código escrito en lenguaje script se incorpora directamente dentro de un código HTML y se ejecuta interpretado. Con la programación del lado del cliente se pueden validar algunos de los datos en la máquina cliente antes de enviarlos al servidor. Esto proporciona a los usuarios informes de error inmediatos, mientras siguen en esa página de formulario y sin necesidad de volver atrás tras recibir un mensaje de error (Puente Cedillo, 2013).

TypeScript 4.4

Es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Desde mediados de 2012. Puede ser usado en el lado del cliente (Angular) o del servidor (Node.js). Mejora la experiencia y la productividad de los desarrolladores. Nos permite utilizar técnicas como el tipado estático opcional y/o la encapsulación para generar un código mucho más mantenible y escalable que con JavaScript tradicional (DOMINICODE, 2019).

Se hará uso de este lenguaje porque permite reutilizar funciones sin importar el tipo de dato con el que se trabaje y además el *framework* para el *frontend* con el que se trabajará está basado en dicho lenguaje.

HTML 5

Es un lenguaje de marcado de elementos y sus siglas significan “*Hyper-Text Markup Language*” (lenguaje de marcado hipertextual). Apela a marcas o etiquetas que brindan información adicional sobre la presentación o la estructura del texto. A diferencia de los lenguajes de programación, los lenguajes de marcado carecen de variables y funciones aritméticas (Definición.de, 2019).

Se decide utilizar ya que permite crear formularios que validan automáticamente los datos ingresados por el usuario.

CSS 3

Las Hojas de estilo en cascada (del inglés *Cascading Stylesheets* CSS) es la siguiente tecnología que aprenderemos después de HTML. Se usa para darle estilo y posicionarlo visualmente. Se puede usar, por ejemplo, para cambiar la fuente, el color, el tamaño y el espaciado del contenido, para formar múltiples columnas, añadir animaciones y otros elementos decorativos (Mozilla, 2020).

Se decide utilizar para dar el estilo que se desea, incluyendo que se pueden utilizar tipos de letras, aunque el usuario no las tenga instaladas y otros efectos visuales.

Lenguaje de programación del lado del servidor

Los lenguajes de programación del lado del servidor son especialmente útiles en trabajos que se tiene que acceder a información centralizada, situada en una base de datos en el servidor, y cuando por razones de seguridad los cálculos no se pueden realizar en la computadora del usuario (Puente Cedillo, 2013).

PHP 8

Es un lenguaje totalmente libre y de código abierto, completamente orientado al desarrollo de aplicaciones web dinámicas y/o páginas web con acceso a una Base de Datos. Posee una curva de aprendizaje muy baja. Es un lenguaje multiplataforma: Windows, Mac OS, Linux e incluso

Unix. Es el lenguaje con mayor usabilidad en el mundo. Posee una versatilidad para la conexión con la mayoría de base de datos que existen en la actualidad (Castillo, 2019).

Se decide trabajar con este lenguaje del lado del servidor ya el *framework* seleccionado para el *backend* está basado en php.

1.5.5 Framework

Es un marco de trabajo, es un conjunto estandarizado y personalizable que facilitan la reutilización de código de elementos comunes usados en el desarrollo web, que facilita y apoya el desarrollo de sitios dinámicos, aplicaciones web y servicios web. Nos permite unificar nuestro trabajo mediante el uso de componentes, plantillas y patrones en común, de tal forma que un proyecto que es desarrollado por una persona bien puede otra distinta dar seguimiento o mantenimiento, ya que cuenta con un esquema uniforme común (RG, 2017).

Angular 12

Es un *framework opensource* desarrollado por Google para facilitar la creación y programación de aplicaciones web de una sola página, las webs SPA (*Single Page Application*). Evita escribir código repetitivo y mantiene todo más ordenado gracias a su patrón MVC (Modelo-Vista-Controlador) asegurando los desarrollos con rapidez, a la vez que posibilita modificaciones y actualizaciones. Es modular y escalable, al estar basado en el estándar de componentes web, y con un conjunto de interfaz de programación de aplicaciones (API) permite crear nuevas etiquetas HTML personalizadas que pueden reutilizarse (Devs, 2019).

Se decide utilizar este *framework* porque al ser basado en componentes permite reutilizarlos fácilmente, además permite que estos sean modificados sin causar daños al resto del código.

Laravel 8

Es un *framework* de PHP, nos ayuda en muchas cosas al desarrollar una aplicación, por medio de su sistema de paquetes y de ser un *framework* del tipo MVC (Modelo-Vista-Controlador) da como resultado que podamos “despreocuparnos” en ciertos aspectos del desarrollo, cómo instanciar clases y métodos para usarlos en muchas partes de nuestra aplicación sin la necesidad de escribirlo y repetirlo muchas veces con lo que eso conlleva a la hora de modificar algo en el código. Funciona desde la línea de comandos para ejecutar muchas funcionalidades como ver todas las rutas de la aplicación disponible, o poner a correr la aplicación o pararla. Es muy potente y sencillo de usar (Altube Vera, 2021).

La herramienta desarrollada el curso anterior utiliza Symfony. En este *framework* la transición a nuevas versiones y los esfuerzos relacionados a los ajustes en el código de la aplicación ya

desarrollada puede ser tedioso. Es de gran complejidad, por lo que es difícil familiarizarse con él.

Se decide sustituir el *framework* anterior por Laravel ya que tiene un entorno de trabajo muy limpio y productivo. Laravel cuenta con una gran comunidad que brinda la posibilidad de corregir errores de formas más rápida. Su motor de plantilla, da numerosas posibilidades para hacer páginas visualmente muy potentes y eficaces, capaz de utilizar sus propias variables y reutilizarlas. Ofrece un nivel bastante fuerte con mecanismos de *hash* y *salt* para encriptar por medio de librerías.

1.5.6 Sistema gestor de base de datos

Un Sistema Gestor de Base de Datos (SGBD) o *DataBase Management System* (DBMS) es un sistema que permite la creación, gestión y administración de bases de datos, así como la elección y manejo de las estructuras necesarias para el almacenamiento y búsqueda de información del modo más eficiente posible (Marín 2019).

PostgreSQL 13

Es un potente sistema de base de datos objeto-relacional. Es de código abierto y gratuito. Se ha ganado una sólida reputación por su arquitectura probada, confiabilidad e integridad de datos, es extensible y multiplataforma (Group 2021). Posee tipos de datos avanzados y permite ejecutar optimizaciones de rendimiento, puede ser instalado en Microsoft Windows, GNU/Linux, MacOS, BSD y muchos otros sistemas operativos. Tiene ahorros considerables en costos de operación.

MySQL 8.0.25

Es un motor de base de datos relacional de código abierto, creado por la empresa MySQL AB, y hace algunos años fue adquirido por la empresa Oracle. De hecho, el programa se puede instalar en cualquier sistema operativo solo o acompañado con otros aplicativos como PHP y phpMyAdmin. Permite crear bases de datos simples o de mediana complejidad con mucha facilidad. Asimismo, es compatible con múltiples plataformas informáticas, y ofrece una infinidad de aplicaciones que ayudan acceder rápidamente a las sentencias del gestor de base de datos. Se destaca por su flexibilidad y su alto rendimiento (Herrera, 2020).

En el desarrollo de la solución se selecciona MySQL por su velocidad a la hora de realizar las operaciones, lo que le hace uno de los gestores que ofrecen mayor rendimiento.

1.5.7 Entorno Integrado de desarrollo

Un entorno de desarrollo integrado (IDE por sus siglas en inglés) es un sistema de software para el diseño de aplicaciones que combina herramientas del desarrollador comunes en una sola interfaz gráfica de usuario (GUI). Los IDE permiten que los desarrolladores comiencen a programar aplicaciones nuevas con rapidez, ya que no necesitan establecer ni integrar manualmente varias herramientas como parte del proceso de configuración (Red Hat, Inc, 2019).

PhpStorm 2021.1.1

Este IDE ofrece edición en directo con tecnologías como CSS, HTML5, JavaScript o *TypeScript* (Ortego Delgado, 2017). El editor realmente 'obtiene' su código y entiende profundamente su estructura, soportando todas las características del lenguaje PHP para proyectos modernos y legados. Proporciona la mejor terminación de código, refactorización, prevención de errores en la marcha y más (JetBrains s.r.o, 2021).

1.5.8 Servidor web

Un servidor web es un ordenador de gran potencia que se encarga de “prestar el servicio” de transmitir la información pedida por sus clientes. Los servidores web son un componente de los servidores que tienen como principal función almacenar, en web hosting, todos los archivos propios de una página web y transmitirlos a los usuarios a través de los navegadores mediante el protocolo Hipertext Transfer Protocol(HTTP) (de Souza, 2019).

Apache 2.4.41

Es un servidor web gratuito y de código abierto, ofrece contenido web a través de Internet. Apache es el servidor web que procesa solicitudes y sirve activos y contenido web a través de HTTP. Tiene una alta capacidad para manejar grandes cantidades de tráfico con una configuración mínima (Hernández, 2019).

Se decide utilizar Apache para el montaje del sistema propuesto ya que con las características que cuenta es uno de los servidores más utilizados, corre en varios sistemas operativos y brinda la posibilidad de hacerle modificaciones de ser necesario.

1.6 Conclusiones del capítulo

En el desarrollo de este capítulo se estudiaron elementos teóricos que sustentan la propuesta de solución del problema planteado, se llegó a la conclusión de que:

- La reutilización es un proceso de suma importancia en el diseño y desarrollo de pruebas, ya que reduce el esfuerzo que se emplea en las mismas.
- El estudio de las técnicas de la reutilización ayudó a comprender mejor el proceso para aplicarlas en la propuesta sin cometer errores.
- El estudio de herramientas homólogas permitió identificar funciones que garantizan la reutilización de los elementos.
- El análisis de las distintas metodologías de desarrollo, herramientas y tecnologías permitió seleccionar las adecuadas para el desarrollo del sistema propuesto.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

En el presente capítulo se describe la herramienta que se desea implementar. Se lleva a cabo el levantamiento de requisitos, los cuales se especifican como funcionales y no funcionales. Se encapsulan los requisitos mediante historias de usuarios como se define en el escenario 4 de la metodología AUP-UCI. Además, se explican los patrones de diseño para la implementación de la herramienta. Se define la arquitectura del software y los estándares de codificación a utilizar, por último, se define el modelo de datos de la base de datos que se empleará.

2.1 Descripción de la herramienta

Como solución se propone implementar un sistema para el diseño de casos de pruebas funcionales en la UCI, empleando las herramientas y tecnologías descritas en el capítulo anterior. Este sistema tiene como objetivo disminuir el esfuerzo empleado en este proceso. Se tendrán en cuenta una serie de requisitos tanto funcionales como no funcionales de acuerdo a las necesidades del cliente y para lograr una mejor interacción con el usuario. Esta herramienta brinda la posibilidad de reutilizar los casos de prueba y escenarios, además de guardarlos como plantilla para usos futuros. Además, la herramienta permite la gestión de los proyectos y da la posibilidad de asociarlos a productos de diferentes tipos.

En la figura 2.1 se muestra el flujo para el diseño de casos de pruebas funcionales:

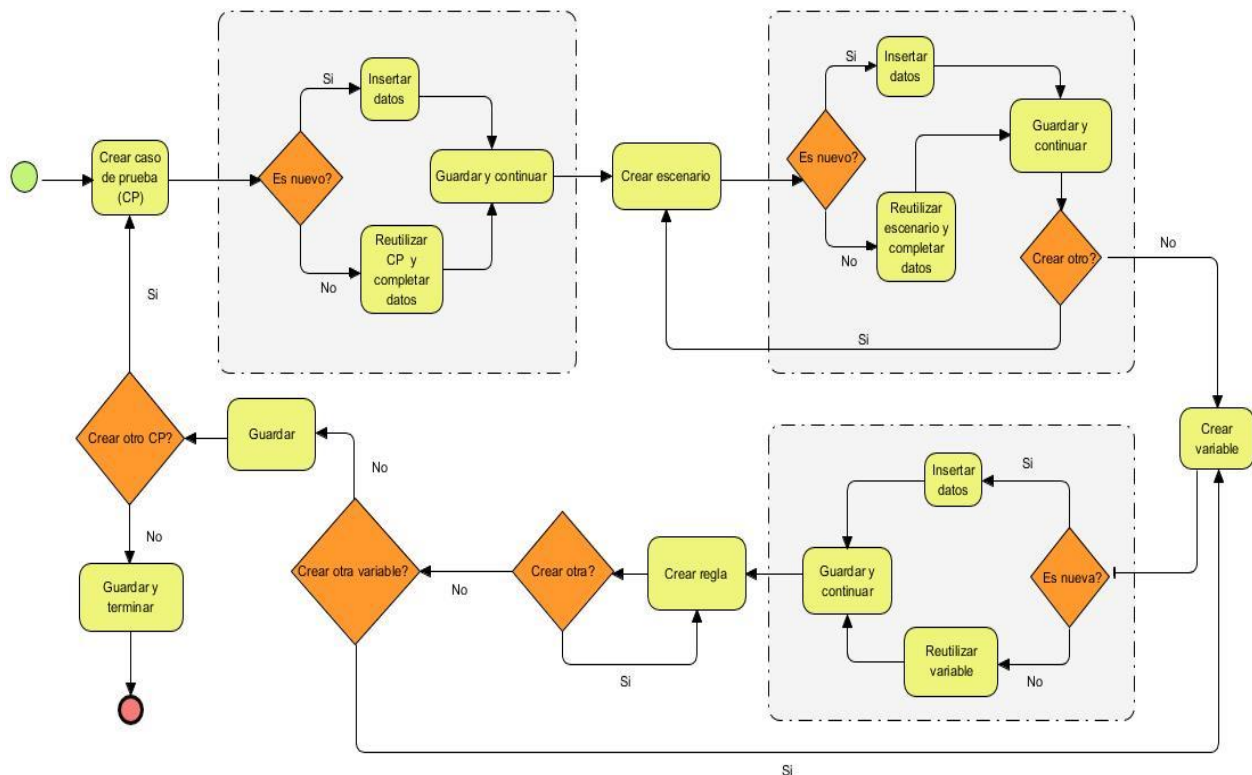


Figura 2. 1: Diseño de casos de pruebas funcionales [Elaboración propia].

El flujo de la figura 2.1 representa el proceso de diseño de casos de pruebas funcionales, aplicando la reutilización de varios elementos. La herramienta que se propone permite la automatización de este flujo, garantizando la aplicación de los conceptos de reutilización presentes en cada paso.



Figura 2. 2: Diseño de casos de pruebas funcionales utilizando la herramienta [Elaboración propia].

En la figura 2.2 se muestra como un usuario con el rol de analista, una vez autenticado en el sistema puede diseñar un CP siguiendo el flujo: CP, escenario, variable y reglas. Para realizar dichas acciones, la aplicación debe brindar la posibilidad de cargar los datos de casos de pruebas y escenarios ya creados con anterioridad para disminuir el tiempo y esfuerzo.

2.2 Definición de requisitos de la herramienta

La ingeniería de requisitos es la fase de un proyecto software donde se definen las propiedades y la estructura del mismo; y que a la vez comprende el desarrollo y gestión de requisitos (Toro Lazo y Gálvez Botero, 2016).

En este epígrafe corresponde el tratado de los requisitos que debe cumplir la herramienta de acuerdo a las necesidades del cliente. Se realizará el levantamiento de los mismos, la especificación en funcionales y no funcionales, además se hará la descripción de los mismos a través del encapsulamiento en historias de usuarios.

La planificación se hará mediante iteraciones, teniendo en cuenta la prioridad de los requisitos a implementar y que al ser un sistema pequeño no se tiene prevista la entrega al cliente por módulos.

En la iteración 1 se llevará a cabo la implementación de los requisitos pertenecientes al módulo administración y la de los requisitos de alta prioridad en el módulo DCP funcionales, para un total de 52 requisitos.

En la iteración 2 se llevará a cabo la implementación de los requisitos que son de prioridad media o baja pertenecientes al módulo DCP funcionales, para un total de 39 requisitos.

En el Anexo D Requisitos correspondientes a cada iteración se muestran los requisitos correspondientes a cada iteración.

2.2.1 Levantamiento de los requisitos

Una de las etapas más importantes dentro del desarrollo de software consiste en la definición de requerimientos, ya que es en esta etapa donde se plasman de forma clara y concisa todos los requisitos del usuario. Una adecuada especificación de requerimientos permitirá generar información consistente, clara y compacta (Rodríguez Barajas, 2018).

En este proceso se pueden emplear varias técnicas como son (Rodríguez Barajas, 2018):

- Análisis de formularios: es una recopilación estructurada de las variables de entrada, a fin de identificar como ingresarán los datos y su recuperación.
- Reúso de requerimientos: los requerimientos capturados con anterioridad pueden usarse nuevamente para especificar una aplicación similar.

Después de realizar un exhaustivo análisis de la “Herramienta para el diseño y gestión de casos de pruebas funcionales” se obtuvo una serie de requisitos necesarios para disminuir el esfuerzo que se emplea en el diseño de casos de prueba. Teniendo en cuenta que algunos requisitos son nuevos, otros se modifican y otros se mantienen suman un total de 84 requisitos funcionales con los cuales debe cumplir el sistema. Se utilizó la plantilla de Especificación de requisitos de software propuesta por el expediente de proyectos de la UCI para especificar cada uno de los requisitos.

2.2.2 Especificación de 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 requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer (Sommerville, 2005). De cada requisito funcional se especifica su descripción, prioridad (P) y complejidad (C).

En la tabla 2.1 se realiza la descripción de cada requisito funcional con la prioridad y la complejidad de cada uno.

Tabla 2. 1: Descripción de requisitos funcionales [Elaboración propia].

N.º	Nombre	Descripción	P	C
Módulo de Administración				
RF1	Registrar usuario	Un usuario puede crearse una cuenta permanente para acceder al sistema.	Alta	Alta
RF2	Autenticar usuario	Permite a los usuarios acceder al sistema mediante usuario y contraseña.	Alta	Alta
RF3	Cerrar sesión	Permite al usuario cerrar su sesión para salir del sistema.	Baja	Baja
RF4	Listar trazas	El administrados y los jefes de proyecto pueden ver las acciones realizadas por los usuarios en el sistema.	Media	Media
RF5	Crear permisos	Permite al usuario con los permisos suficientes registrar un permiso en el sistema.	Alta	Media
RF6	Modificar permisos	Permite al usuario con los permisos suficientes modificar los datos de un permiso registrado en el sistema.	Media	Media
RF7	Listar permisos	Permite al usuario con los permisos suficientes ver el listado de los permisos registrados en el sistema.	Media	Media
RF8	Eliminar permisos	Permite al usuario con los permisos suficientes eliminar un permiso del sistema.	Baja	Media
RF9	Mostrar permisos	Permite al usuario con los permisos suficientes ver los datos de un permiso en específico registrado en el sistema y los roles a los que se le asignó ese permiso.	Media	Media
RF10	Asignar permisos	Permite al usuario con los permisos suficientes asignar un permiso a un rol registrado en el sistema.	Alta	Media
RF11	Quitar permisos	Permite al usuario con los permisos suficientes quitar un permiso a un rol registrado en el sistema.	Media	Media

RF12	Crear usuario	Los administradores podrán crear usuarios dentro del sistema.	Media	Media
RF13	Modificar usuario	Luego de autenticado un usuario, el sistema permite que este pueda modificar sus datos de usuario.	Media	Media
RF14	Listar usuario	El sistema permite hacer un listado de todos los usuarios registrados en él.	Media	Baja
RF15	Mostrar usuario	Permite al usuario con los permisos suficientes ver la información de otro usuario o los datos del propio usuario.	Baja	Media
RF16	Eliminar usuario	Permite al usuario con los permisos suficientes eliminar a otro usuario registrado en el sistema.	Baja	Baja
RF17	Activar usuario	Permite al administrador y a los jefes de proyecto activar un usuario en el sistema.	Media	Media
RF18	Desactivar usuario	Permite al administrador y a los jefes de proyecto desactivar un usuario en el sistema.	Media	Media
RF19	Crear rol	El sistema permite al administrador crear roles.	Alta	Media
RF20	Modificar rol	Permite al usuario con los permisos suficientes modificar un rol registrado antes en el sistema.	Media	Baja
RF21	Listar rol	Lista todos los roles creados en el sistema.	Media	Baja
RF22	Mostrar rol	Permite al usuario con los permisos suficientes ver los detalles de un rol.	Media	Media
RF23	Eliminar rol	Un rol puede ser eliminado si no es de utilidad.	Baja	Media
RF24	Asignar rol	Permite asignar un rol a un usuario registrado en el sistema.	Media	Media
RF25	Quitar rol	Permite quitar un rol a un usuario registrado en el sistema.	Media	Media
RF26	Crear proyecto	Permite al administrador y al jefe de proyecto crear un proyecto en el sistema.	Alta	Media
RF27	Modificar proyecto	Permite al administrador y al jefe de proyecto modificar un proyecto en el sistema.	Media	Baja
RF28	Listar proyecto	Lista todos los usuarios registrados en el sistema.	Media	Baja

RF29	Mostrar proyecto	Permite a los usuarios con los permisos necesarios ver los detalles del proyecto.	Media	Media
RF30	Eliminar proyecto	Permite eliminar un proyecto que ya no esté en uso.	Media	Media
RF31	Añadir usuario	Permite añadir un usuario a un proyecto.	Media	Media
RF32	Crear producto	El sistema permite que el usuario con rol jefe de proyecto cree los productos.	Alta	Media
RF33	Modificar producto	El sistema permite que el usuario con rol jefe de proyecto modifique los productos creados.	Media	Media
RF34	Listar producto	El sistema permite que el usuario con rol jefe de proyecto liste los productos creados.	Media	Media
RF35	Mostrar producto	Permite a los usuarios con los permisos necesarios ver los detalles del producto.	Media	Media
RF36	Eliminar producto	El sistema permite que el usuario con rol jefe de proyecto elimine productos.	Media	Media
RF37	Crear tipo de producto	El sistema permite que el administrador cree tipos de productos.	Alta	Media
RF38	Modificar tipo de producto	El sistema permite que el administrador modifique los tipos de Productos.	Media	Media
RF39	Listar tipo de producto	El sistema permite que el administrador liste los tipos de productos.	Media	Media
RF40	Eliminar tipo de producto	El sistema permite que el administrador elimine los tipos de productos.	Media	Media
RF41	Crear centro de desarrollo	El sistema permite que el administrador cree centros de desarrollo.	Alta	Media
RF42	Modificar centro de desarrollo	El sistema permite que el administrador modifique centros de desarrollo.	Media	Media
RF43	Listar centro de desarrollo	El sistema permite que el administrador liste los centros de desarrollo.	Media	Media
RF44	Mostrar centro de desarrollo	Permite a los usuarios con los permisos necesarios ver los detalles del centro de desarrollo.	Media	Media

RF45	Eliminar centro de desarrollo	El sistema permite que el administrador elimine centros de desarrollo.	Media	Media
Módulo de Diseño de Casos de Prueba Funcionales				
RF46	Crear particiones de un campo	Un usuario puede crear particiones de variables ya introducidas en el sistema.	Alta	Alta
RF47	Modificar particiones de un campo	El sistema permite que los usuarios puedan modificar las particiones creadas a las variables.	Media	Media
RF48	Listar particiones de un campo	Un usuario puede consultar el listado de particiones creadas.	Media	Media
RF49	Eliminar particiones de un campo	El sistema permite que un usuario pueda eliminar particiones creadas a las variables.	Media	Media
RF50	Crear casos de prueba	El usuario tendrá la opción de crear un nuevo caso de prueba luego de haber creado un proyecto.	Alta	Alta
RF51	Cargar casos de prueba	El usuario podrá cargar un caso de prueba para reutilizar sus datos.	Alta	Alta
RF52	Modificar casos de prueba	El sistema permite al usuario modificar un caso de prueba de un proceso que ha creado.	Media	Media
RF53	Listar casos de prueba	Un usuario puede consultar el listado de casos de prueba creados.	Media	Media
RF54	Mostrar casos de prueba	Muestra los detalles de un caso de prueba seleccionado.	Media	Media
RF55	Eliminar casos de prueba	Luego de autenticado un usuario, el sistema permite que dicho usuario elimine un caso de prueba de un proceso que ha creado.	Alta	Alta
RF56	Exportar casos de prueba	El usuario tendrá la opción de exportar los casos de prueba con toda su información.	Alta	Alta
RF57	Cargar casos de pruebas plantillas	El usuario podrá cargar un caso de prueba plantilla para reutilizar sus datos.	Alta	Alta
RF58	Modificar casos de prueba plantilla	Modifica los datos de una plantilla de caso de prueba específica.	Media	Media

RF59	Listar casos de prueba plantilla	Lista todas las plantillas de casos de prueba creados en el sistema.	Media	Media
RF60	Exportar casos de prueba plantilla	El usuario tendrá la opción de exportar los casos de prueba con toda su información.	Alta	Alta
RF61	Eliminar casos de prueba plantilla	La plantilla de un caso de prueba podrá ser eliminada si no es de utilidad.	Media	Media
RF62	Guardar casos de prueba como plantilla	El usuario podrá guardar un caso de prueba como plantilla para su posterior uso.	Alta	Media
RF63	Crear escenario	El usuario tendrá la opción de crear escenarios en un CP.	Alta	Alta
RF64	Cargar escenario	El usuario podrá cargar un escenario para reutilizar sus datos.	Alta	Alta
RF65	Modificar escenario	El sistema permite al usuario modificar un escenario que ha creado.	Media	Media
RF66	Listar escenario	Un usuario puede listar los escenarios que ha creado.	Media	Media
RF67	Mostrar escenario	Muestra los detalles de un determinado escenario.	Media	Media
RF68	Eliminar escenario	Luego de autenticado un usuario, el sistema permite que dicho usuario elimine un escenario que ha creado.	Media	Media
RF69	Cargar escenario plantilla	El usuario podrá cargar un escenario plantilla para reutilizar sus datos.	Alta	Alta
RF70	Modificar escenario plantilla	Modifica los datos que contiene la plantilla de un escenario.	Media	Media
RF71	Listar escenario plantilla	Lista todas las plantillas de escenarios creadas en el sistema.	Media	Media
RF72	Mostrar escenario plantilla	Permite a los usuarios ver los detalles de una plantilla de escenario.	Baja	Media
RF73	Eliminar escenario plantilla	La plantilla de un escenario podrá ser eliminada si no es de utilidad.	Media	Media

RF74	Guardar escenario como plantilla	El usuario podrá guardar un escenario como plantilla para su posterior uso.	Alta	Media
RF75	Crear variable	El usuario tendrá la opción de crear variables para un escenario.	Alta	Alta
RF76	Cargar variable	El usuario podrá cargar una o más variables.	Alta	Alta
RF77	Modificar variable	El sistema permite al usuario modificar una variable que ha creado.	Media	Media
RF78	Listar variable	Un usuario puede listar las variables que ha creado.	Media	Media
RF79	Mostrar variable	Muestra los detalles de una variable creada en el sistema.	Media	Media
RF80	Eliminar variable	Luego de autenticado un usuario, el sistema permite que dicho usuario elimine una variable que ha creado.	Media	Media
RF82	Crear regla	Un usuario puede crear reglas para que luego sean asociadas a las variables.	Media	Media
RF83	Listar regla	Un usuario puede eliminar reglas creadas.	Media	Media
RF84	Eliminar regla	Permite a los usuarios eliminar reglas creadas.	Media	Media

2.2.3 Especificación de 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 requerimientos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente apenas se aplican a características o servicios individuales del sistema (Sommerville, 2005).

En la tabla 2.2 se muestran los requisitos no funcionales con los que debe cumplir el sistema.

Tabla 2. 2: Descripción de requisitos no funcionales [Elaboración propia].

N.º	Descripción
Seguridad	
RnF1	Los usuarios tendrán acceso a las funcionalidades del sistema de acuerdo al rol que tengan asignado como se observa en la tabla 2.3.
RnF2	El registro de nuevos usuarios debe terminar con la verificación del correo electrónico.
RnF3	Existirá un solo administrador del sistema
RnF4	El acceso al sistema se realizará mediante un proceso de autenticación

RnF5	La contraseña del usuario debe tener un mínimo de 8 caracteres, combinación de letras, números y caracteres especiales.
Rendimiento	
RnF6	El proceso de ejecución de la prueba tendrá prioridad en cuanto a la asignación de recursos.
RnF7	El sistema debe permitir una concurrencia no menor de 100 usuarios
RnF8	El sistema debe permitir la ejecución simultánea de al menos 10 pruebas
RnF9	Cada transición demorará, como promedio, cinco segundos.
Portabilidad	
RnF10	Se podrá acceder a la aplicación desde los navegadores web Mozilla Firefox y Chrome.
Confiabilidad	
RnF11	El sistema establecerá mecanismos para garantizar la confiabilidad e integridad de la información ante posibles accesos no autorizados.
RnF12	Tendrá implementado mecanismos de tratamientos de errores.
Soporte	
RnF13	Permitirá la incorporación de nuevas funcionalidades en caso de ser necesarias.
Usabilidad	
RnF14	Cada campo debe tener un identificador intuitivo.
RnF15	Debe existir un correcto contraste en los colores de la aplicación.
RnF16	El sistema debe mantener un diseño coherente en toda la web y sus distintos enlaces.

En la tabla 2.3 que se muestra a continuación se realiza la asignación de permisos por cada rol del sistema.

Tabla 2. 3: Asignación de permisos por roles [Elaboración propia].

Rol	Permisos
Super Administrador	Módulo administración (RF1, RF2, RF3, RF4, RF5, RF6, RF7, RF8, RF9, RF10, RF11, RF12, RF13, RF14, RF15, RF16, RF17, RF18, RF19, RF20, RF21, RF22, RF23, RF24, RF25, RF26, RF27, RF28, RF29, RF30, RF31, RF32, RF33, RF34, RF35, RF36, RF37, RF38, RF39, RF40, RF41, RF42, RF43, RF44, RF45) Módulo diseño de casos de pruebas funcionales (RF46, RF47, RF48, R49, RF50, RF51, RF52, RF53, RF54, RF55, RF56, RF57, RF58, RF59, RF60,

	RF61, RF62, RF63, RF64, RF65, RF66, RF67, RF68, RF69, RF70, RF71, RF72, RF73, RF74, RF75, RF76, RF77, RF78, RF79, RF80, RF81, RF82, RF83, RF84, RF85)
Administrador	Módulo administración (RF1, RF2, RF3, RF4, RF5, RF6, RF7, RF8, RF9, RF10, RF11, RF12, RF13, RF14, RF15, RF16, RF17, RF18, RF19, RF20, RF21, RF22, RF23, RF24, RF25, RF26, RF27, RF28, RF29, RF30, RF36, RF37, RF38, RF39, RF40, RF41, RF42, RF43, RF44, RF45)
Jefe de Proyecto	Módulo administración (RF1, RF2, RF3, RF4, RF5, RF13, RF14, RF15, RF16, RF17, RF18, RF19, RF26, RF27, RF28, RF29, RF30, RF31, RF32, RF33, RF34, RF35) Módulo diseño de casos de pruebas funcionales (RF47, RF52, RF53, RF55, RF58, RF59, RF65, RF66, RF70, RF71, RF77, RF78, RF82)
Analista	Módulo administración (RF1, RF2, RF3, RF4) Módulo diseño de casos de pruebas funcionales (RF46, RF47, RF48, R49, RF50, RF51, RF52, RF53, RF54, RF55, RF56, RF57, RF58, RF59, RF60, RF61, RF62, RF63, RF64, RF65, RF66, RF67, RF68, RF69, RF70, RF71, RF72, RF73, RF74, RF75, RF76, RF77, RF78, RF79, RF80, RF81, RF82, RF83, RF84, RF85)
Coordinador de pruebas	Módulo administración (RF1, RF2, RF3, RF4) Módulo diseño de casos de pruebas funcionales (RF46, RF47, RF48, R49, RF50, RF51, RF52, RF53, RF54, RF55, RF56, RF57, RF58, RF59, RF60, RF61, RF62, RF63, RF64, RF65, RF66, RF67, RF68, RF69, RF70, RF71, RF72, RF73, RF74, RF75, RF76, RF77, RF78, RF79, RF80, RF81, RF82, RF83, RF84, RF85)
Probador	Módulo administración (RF1, RF2, RF3, RF4) Módulo diseño de casos de pruebas funcionales (RF46, RF47, RF48, R49, RF50, RF51, RF52, RF53, RF54, RF55, RF56, RF57, RF58, RF59, RF60, RF61, RF62, RF63, RF64, RF65, RF66, RF67, RF68, RF69, RF70, RF71, RF72, RF73, RF74, RF75, RF76, RF77, RF78, RF79, RF80, RF81, RF82, RF83, RF84, RF85)

2.2.4 Descripción de requisitos mediante HU

Las historias de usuario son una herramienta que agiliza la administración de requisitos, reduciendo la cantidad de documentos formales y tiempo necesarios. Se usan, en el contexto de

la ingeniería de requisitos ágil, como una herramienta de comunicación que combina las fortalezas de ambos medios: escrito y verbal (Menzinsky, 2020).

Para el diseño de este sistema se seleccionó el escenario 4 de la metodología AUP-UCI, el cual encapsula los requisitos funcionales del sistema mediante el uso de HU. Se hace una descripción de lo que se espera una vez implementado el requisito y el beneficio que aporta al usuario. En las tablas 2.4, 2.5 y 2.6 se mostrarán las historias de usuarios de nuevos requisitos:

Tabla 2. 4: Historia de usuario #1 [Elaboración propia]

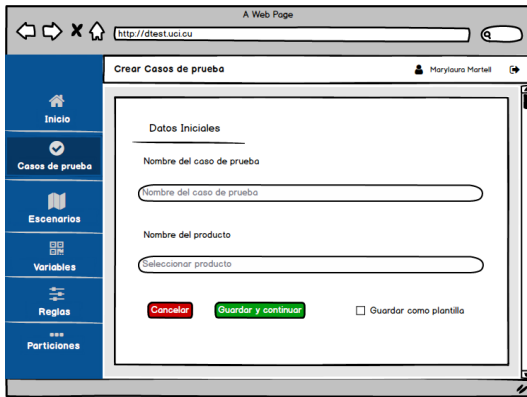
Historia de Usuario	
Número: 1	Usuario: Cliente
Nombre de historia: Guardar casos de prueba como plantilla	
Prioridad en negocio: Alta	Riesgo en desarrollo: Medio
Puntos estimados:	Iteración asignada: 2
Programador Responsable: Marylaura Martell León	
<p>Descripción:</p> <p>Una vez autenticado el usuario, en el menú que aparece a la izquierda de la vista de la herramienta, hacer clic en la opción Casos de prueba, se desplegará el menú donde se observa la opción Crear caso de prueba en la cual debe hacer clic. La aplicación muestra todos los datos que deben ser llenados para poder crear un CP.</p> <p>Luego de llenados al menos todos los campos obligatorios, debe tocar activar la opción de guardar como plantilla que se encuentra al final del formulario.</p>	
<p>Validación:</p> <p>El usuario puede guardar un caso de prueba como plantilla para su posterior uso.</p>	
<p>Interfaz</p> 	

Tabla 2. 5: Historia de usuario #2 [Elaboración propia]

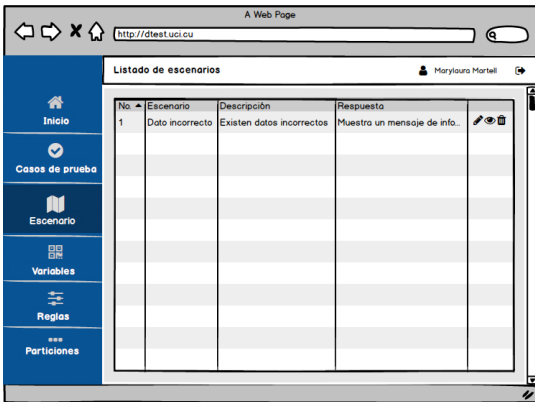
Historia de Usuario	
Número: 2	Usuario: Cliente
Nombre de historia: Mostrar escenario	
Prioridad en negocio: Media	Riesgo en desarrollo: Medio
Puntos estimados:	Iteración asignada: 2
Programador Responsable: Marylaura Martell León	
<p>Descripción:</p> <p>Una vez autenticado el usuario, en el menú que aparece a la izquierda de la vista de la herramienta, hacer clic en la opción Escenario, se desplegará el menú donde se observa la opción Listado de escenarios en la cual debe hacer clic. La aplicación muestra una tabla que contiene todos los escenarios creado en el sistema.</p> <p>En la última columna de la tabla debe seleccionar la opción de Mostrar escenario de acuerdo al que necesite ver, después de realizada esta acción se abrirá una vista donde se muestran todos los detalles del escenario seleccionado.</p>	
<p>Validación:</p> <p>El usuario puede ver los detalles de un escenario.</p>	
<p>Interfaz</p> 	

Tabla 2. 6: Historia de usuario #3 [Elaboración propia]

Historia de Usuario	
Número: 3	Usuario: Cliente
Nombre de historia: Mostrar variable	
Prioridad en negocio: Media	Riesgo en desarrollo: Medio
Puntos estimados:	Iteración asignada: 2

Programador Responsable: Marylaura Martell León

Descripción:

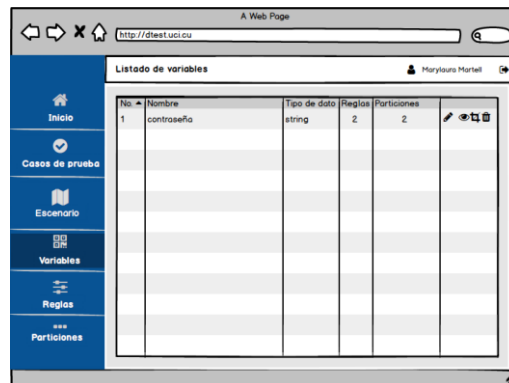
Una vez autenticado el usuario, en el menú que aparece a la izquierda de la vista de la herramienta, hacer clic en la opción Variable, se desplegará el menú donde se observa la opción Listado de variables en la cual debe hacer clic. La aplicación muestra una tabla que contiene todas las variables creadas en el sistema.

En la última columna de la tabla debe seleccionar la opción de Mostrar variable de acuerdo a la que necesite ver, después de realizada esta acción se abrirá una vista donde se muestran todos los detalles de la variable seleccionada.

Validación:

El usuario puede ver los detalles de una variable.

Interfaz



Estimación de Historias de usuarios

La estimación de historias de usuarios es la aproximación del esfuerzo necesario (en tiempo ideal) para implementarla. Puede estimarse usando unidades de desarrollo (puntos de historia), si el equipo lo prefiere y está familiarizado con este sistema (Menzinsky, López, Palacio, Sobrino, Álvarez y Rivas, 2020).

De acuerdo a que la estimación está basada en la información que se tiene en el momento que se realiza, puede que sea necesario reajustar esas estimaciones cuando haya nueva información, para lograr un aproximado más exacto del tiempo que llevará cada iteración.

En casos donde se pueda estimar la tarea de manera individual por tener una buena experiencia es recomendable utilizar la técnica estimación por dos puntos. En esta técnica la estimación es ascendente y tiene una alta probabilidad de que el valor real este entre el mejor y el peor caso.

En la figura 2.2 se muestra la fórmula que propone esta técnica:

$$\text{Resultado} = \frac{\text{MejorCaso} + \text{PeorCaso}}{2}$$

Figura 2. 3: Estimación por dos puntos (Müller y Friedenber, 2011).

Tabla 2. 7: Estimación de dos puntos por HU [Elaboración propia]

Historia de usuario	Mejor caso	Peor caso	Resultado
Registrar usuario	4 horas	6 horas	5 horas
Autenticar usuario	3 horas	5 horas	4 horas

El desarrollo de la propuesta involucra un desarrollador, considerando esto y teniendo en cuenta la información de la tabla 2.7, se lleva a cabo la planificación del tiempo aproximado de duración de las iteraciones para la implementación de la siguiente manera:

Iteración 1: 217 horas

Iteración 2: 156 horas

Fueron identificadas 78 HU, de ellas 32 fueron actualizadas de acuerdo a su descripción con el objetivo de mejorar el entendimiento de las mismas.

2.3 Diseño de la propuesta de solución

La metodología de desarrollo de software seleccionada es la AUP-UCI, la cual sugiere que se realicen diseños que sean más fácil de entender a la hora de implementarlos, logrando de esta manera emplear menor tiempo en el desempeño de esta tarea.

2.3.1 Arquitectura de software

Los *frameworks* Angular y Laravel utilizan la arquitectura Modelo Vista Controlador (MVC), el cual es un enfoque de software que separa la lógica de una aplicación, de la presentación o interfaz de usuario. Se divide en tres capas (Ramírez, 2017):

- El modelo representa las estructuras de datos, contiene las funciones que ayudan a mostrar, insertar, actualizar y eliminar información de la base de datos.
- La vista permite al usuario interactuar con la información a través de una página web.
- El controlador funciona como intermediario entre el modelo, la vista y cualquier otro recurso necesario para procesar una solicitud HTTP, y con esto, poner a funcionar una página.

En la figura 2.1 se muestra el funcionamiento del MVC.

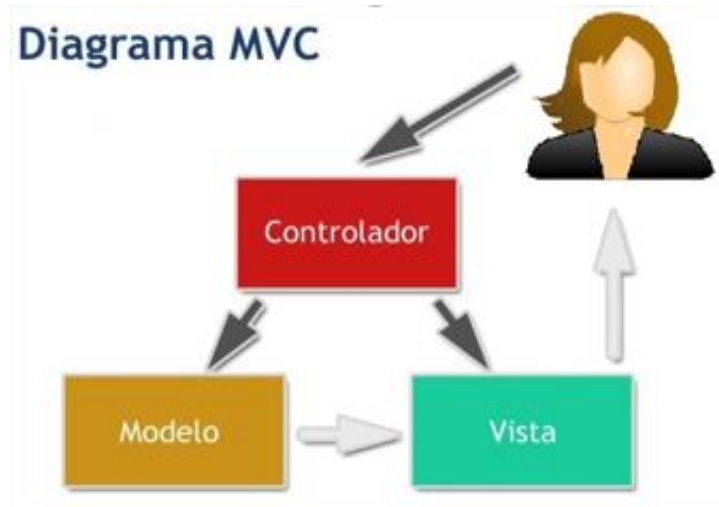


Figura 2. 4: Funcionamiento de la arquitectura MVC (Ramírez, 2017).

Con el empleo de la arquitectura MVC se separa la lógica del negocio y la presentación de la información, con ello se logra crear un software más ordenado y permite potenciar su mantenimiento y reutilización de código con mayor facilidad. Con este patrón se pueden realizar proyectos pequeños o de gran escala. Brinda la posibilidad de cambiar la vista de acuerdo al dispositivo que se utilice sin afectar el modelo creado originalmente, utilizando al controlador como intermediario. La figura 2.1 muestra el funcionamiento de la arquitectura descrita.

2.3.2 Patrones de diseño

De acuerdo al análisis realizado por (Castañeda Rojas, 2016) un patrón de diseño describe problemas comunes de software y detalla la solución de este, la cual debe ser reutilizable. Son descripciones de comunicación entre objetos y clases que pueden adaptarse para resolver un problema de diseño general en un contexto particular.

Los Patrones Generales de Software para la Asignación de Responsabilidades (GRASP) son utilizados para describir los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones (Larman, 2003).

- **Experto:** con este patrón se mantiene el encapsulamiento de la información en una clase a la cual le será asignada toda la responsabilidad de creación de objetos e implementación de métodos acorde a la lógica del negocio que le corresponda.
- **Controlador:** el patrón controlador actúa como intermediario entre una interfaz y el algoritmo que la implementa, es el encargado de controlar el flujo de datos de las funcionalidades que se le fueron asignadas y los eventos asociados a estas.
- **Alta Cohesión:** la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase y al objetivo para lo cual fue

creada. Con la Cohesión Lógica el módulo realiza múltiples tareas relacionadas, pero, en tiempo de ejecución, sólo una de ellas será llevada a cabo.

- **Bajo Acoplamiento:** la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

2.3.3 Estándares de codificación

Los estándares de codificación son un conjunto de reglas que definen la apariencia del código y su estructura, facilitando que sea más legible y se pueda dar mantenimiento de forma más sencilla. Angular define como estilo de código el Camel y Laravel define como estándar de código PSR-1 y PSR-4, por lo que en el desarrollo de la propuesta se sigue la siguiente estructura de código:

- Las variables deben ser descriptivas.
- Utilizar tipo “*service*” para identificar servicios.
- Utilizar tipo “*component*” para identificar componentes.
- Utilizar tipo “*module*” para identificar módulos.
- Poner sufijo de acuerdo a su tipo.
- Emplear “-” para separar palabras en nombres descriptivos.
- Emplear “.” Para separar el nombre descriptivo del tipo.
- Los nombres de clases comienzan con mayúscula y de tener más de una palabra comenzando cada una con mayúscula.
- Los nombres de funciones empiezan con minúscula y de tener más de una palabra comenzando cada una con mayúscula.
- Usar siempre el *tag* de apertura largo de PHP `<?php`.
- Declarar las propiedades de las clases antes de los métodos.
- Declarar los métodos en este orden: *public*, *protected* y *private*.
- Usar los operadores lógicos `&&` y `||` en lugar de *AND* y *OR*.

En el desarrollo de la propuesta de solución se emplearon diferentes estilos que se muestran a continuación:

Definición de clases:

El nombre de la clase tiene más de una palabra, todas empiezan con mayúscula y la llave de apertura se encuentra al final de la línea.

```
export class FormularioCasoPruebaComponent implements OnInit {  
  
  constructor() {  
  }  
  
  ngOnInit(): void {  
  }  
  
}
```

Código fuente 2. 1: Definición de clases

Definición de métodos:

La declaración del método comienza con las palabras *public*, *private* o *protected* seguido del nombre, este comienza en minúscula y si tiene más de una palabra las demás comienzan en mayúscula.

```
public guardarCasoPrueba(): void {  
  // ...  
}
```

Código fuente 2. 2: Definición de métodos

Declaración de variables:

Las variables se pueden declarar sin inicializar poniendo la palabra *let* o *const* seguido del nombre y el tipo de datos. Se puede declarar y en vez de poner el tipo de dato se iguala al valor con el que se desea iniciar la variable.

```
let nombreCP: string;
```

Código fuente 2. 3: Declaración de variable sin inicializar.

```
let nombreCP = 'Caso de prueba';
```

Código fuente 2. 4: Declaración de variable inicializada

2.3.4 Modelo de datos

Una base de datos es utilizada para guardar toda la información de un sistema determinado para su posterior uso.

El modelo entidad relación representa la relación entre dos o más entidades, ayuda a construir una estructura lógica de base de datos (Edrawsoft, 2021).

El modelo cuenta con 20 tablas y 19 relaciones. Las relaciones del modelo están definidas de acuerdo a las funcionalidades que el mismo requiere. Un usuario puede tener un rol con el cual accede al sistema y este rol a su vez tiene permisos asignados. Los casos de prueba pueden tener muchos escenarios, los cuales pueden tener muchas variables, además un escenario se puede utilizar en varios casos de prueba.

En la figura 2.2 se muestra el modelo de datos generado:

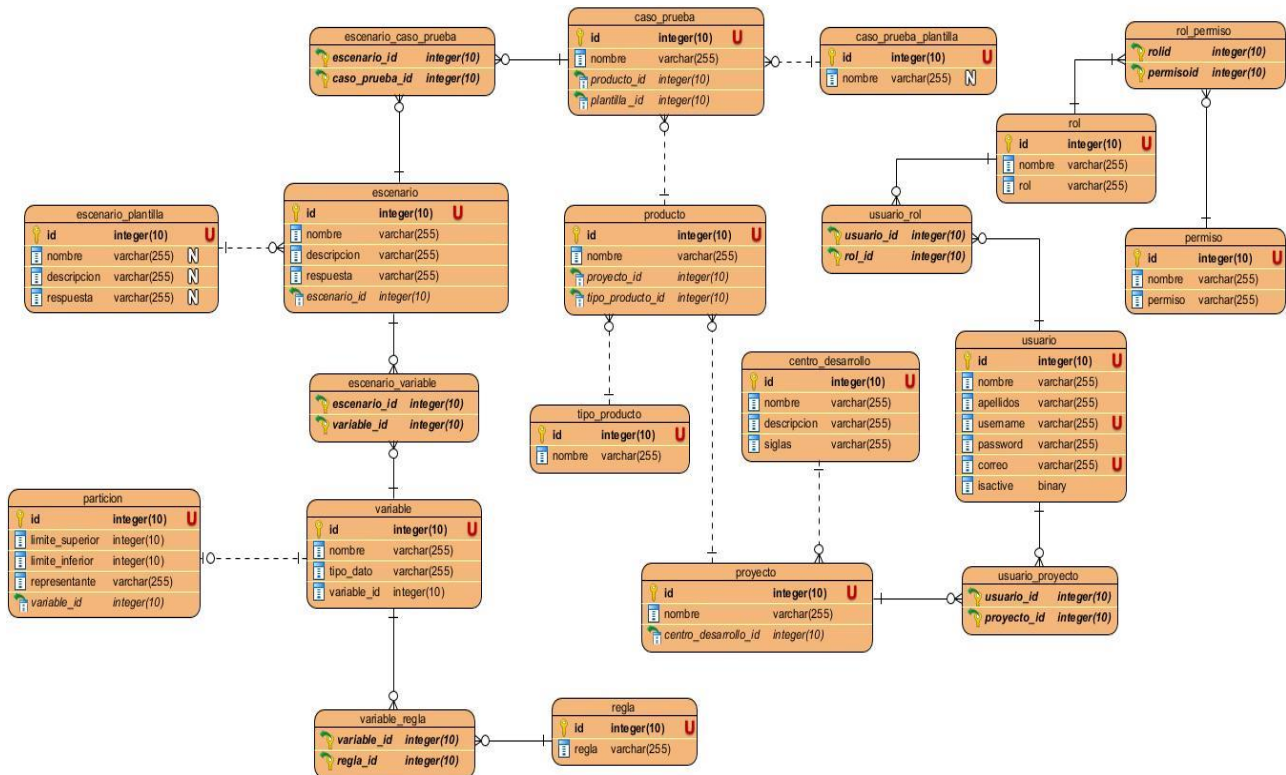


Figura 2. 5: Modelo de datos [Elaboración propia].

El modelo representado en la figura 2.3 garantiza el uso adecuado de las buenas prácticas de la reutilización. Con un buen modelo, los datos almacenados no se verán expuestos a continuas transformaciones ya que cuentan con una estructura que reflejan entidades del mundo real.

2.4 Conclusiones del capítulo

Luego de definir las características del sistema se puede llegar a las siguientes conclusiones:

- El análisis de la “Herramienta para el diseño y gestión de casos de pruebas funcionales” fue de gran utilidad para identificar las funcionalidades que se necesita en la propuesta y las restricciones que debe cumplir. Los requisitos obtenidos se encapsularon en HU.
- Fue necesario realizar una estimación por HU para saber el tiempo y esfuerzo que puede llegar a necesitar el desarrollo de la propuesta.

- Se identificó el estilo arquitectónico y los patrones de diseño a utilizar en la propuesta lo que permite realizar cambios futuros en el código sin generar gran impacto.
- Se generó el modelo de datos con el que contará el sistema de acuerdo a todas las funcionalidades y restricciones del mismo.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DEL SISTEMA

Luego de realizar el análisis y diseño del sistema, corresponde en este capítulo obtener los resultados de acuerdo a las disciplinas de implementación y pruebas del software. Se diseñan y aplican las pruebas para comprobar el correcto funcionamiento de la herramienta. Se muestran las conexiones del hardware y se realiza la validación de la propuesta.

3.1 Implementación de la propuesta de solución

En esta disciplina se lleva a cabo la implementación de las HU planificadas para cada iteración. El plan de iteraciones se revisa luego de concluir cada iteración con el objetivo de ajustarlo a los cambios aprobados. Para llevar a cabo la implementación de las HU se asignan tareas al equipo o persona que desarrollará la propuesta. Se debe emplear un lenguaje técnico ya que esta actividad cuenta con la participación de los desarrolladores.

Iteración 1: En esta iteración se implementaron 53 de las 53 HU planificadas para la misma.

Iteración 2: En esta iteración se implementaron 38 de las 38 HU planificadas para la misma.

A continuación, en la tabla 3.1, puede observarse el tiempo real que se empleó en la implementación de algunas de las HU de ambas iteraciones.

Tabla 3. 1: Tiempo de implementación de las HU de ambas iteraciones [Elaboración propia].

Historia de usuario	Tiempo estimado	Tiempo Real	Iteración
Registrar usuario	5 horas	3 horas	1
Autenticar usuario	4 horas	2 horas	1
Crear proyecto	4 horas	3 horas	1
Listar casos de prueba plantilla	3 horas	2 horas	2
Guardar casos de prueba como plantilla	5 horas	4 horas	1 y 2
Guardar escenario como plantilla	5 horas	4 horas	1 y 2

3.2 Diagrama de despliegue

El diagrama de despliegue muestra la configuración de los elementos de hardware (nodos) y la disposición física de los artefactos software en esos nodos. Es considerado necesario realizar este diagrama para modelar el hardware utilizado en la implementación del sistema y las relaciones entre sus componentes (Caicedo Salazar, Guerrero Arellano y Pombar Vallejos, 2017).

El diagrama de despliegue se utiliza para visualizar dispositivos de hardware de un sistema, los enlaces de comunicación entre ellos y la colocación de los archivos de software en ese hardware. Muestra la arquitectura de ejecución de un sistema.

En la figura 3.1 que se muestra el diagrama de despliegue que corresponde con la herramienta que se desea implementar.

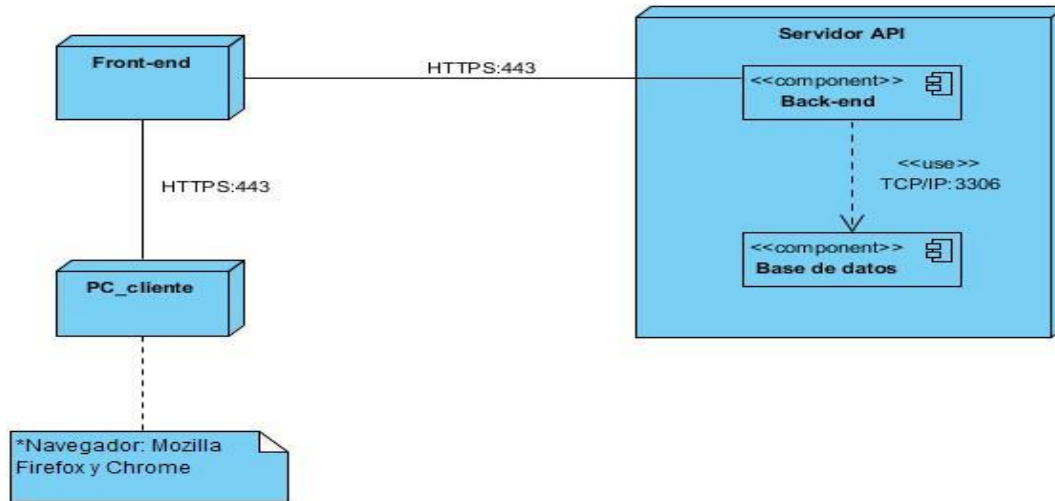


Figura 3. 1: Diagrama de despliegue [Elaboración propia].

En el diagrama anterior se muestra las distintas conexiones entre el hardware que se adecua a la herramienta que se desea implementar. El servidor se compone del *back-end* de la aplicación y la base de datos de la misma donde se almacena toda la información del sistema.

3.3 Pruebas de software

La prueba de software es un proceso, o una serie de procesos, diseñado para asegurarse de que el código de la computadora haga lo que fue diseñado para hacer y que no hace nada involuntario. El software debe ser predecible y coherente, sin sorpresas para los usuarios. Las pruebas de software, se definen como el proceso de aplicar unos pocos criterios de prueba de propósito general bien definidos a una estructura o modelo del software (Kumar Singh y Singh, 2016).

3.3.1 Estrategia de pruebas

Para la ejecución de las pruebas se emplearán los productos de trabajo donde se describió toda la información necesaria durante el desarrollo de la herramienta. Estos productos son:

- Especificación de requisitos
- Historias de Usuario

Para describir los tipos de pruebas que se le realizarán al sistema se tuvo en cuenta los requisitos adquiridos durante el diseño de la propuesta y las características de calidad identificadas en la norma (ISO/IEC 25010, 2011). En la tabla 3.2 se muestran los tipos de pruebas que aparecen en (DCS):

Tabla 3. 2: Tipos de prueba del sistema (DCS).

Tipos de prueba	Descripción
Funcionalidad	Se basa en el chequeo de los requisitos definidos durante el diseño de acuerdo a las funcionalidades presentes en el sistema.
Usabilidad	Se analiza el comportamiento de usuarios mientras realizan tareas en el sistema.
Seguridad	Analiza la seguridad del sistema en cuanto al acceso al mismo y a la información de personas autorizadas.
Portabilidad	Verifica que el producto pueda ser utilizado en diferentes entornos de hardware, software u otros.
Confiabilidad	Se encarga de verificar que los procesos de recuperación llevan al estado deseado y a su vez mantiene informados a los usuarios.
Mantenibilidad	Se analiza la facilidad con la que el producto puede ser modificado de forma efectiva debido a necesidades correctivas o evolutivas.

Las listas de chequeo son formatos generados para realizar actividades repetitivas, controlar el cumplimiento de un listado de requisitos o recolectar datos ordenadamente y de manera sistemática. Se utilizan para hacer comprobaciones sistemáticas de actividades o productos (ISOTools, 2018).

Teniendo esto en cuenta fue necesario utilizar una lista de chequeo con el objetivo de analizar y comprobar que se cumplieran correctamente cada uno de los requisitos no funcionales con los que debe contar el sistema. Las pruebas a los requisitos de rendimiento no mostrarán resultados como parte de la investigación ya que serán realizadas en el entorno real de la aplicación.

Con el objetivo de comprobar que se solucionaron sin problema los defectos encontrados en la primera iteración, se decidió realizar pruebas de regresión al comienzo de la segunda iteración planificada.

Los roles que intervinieron en el desarrollo de las pruebas son:

- Jefe de equipo
- Coordinador de prueba
- Desarrollador
- Probador

En la tabla 3.3 se muestra el cronograma de planificación de las pruebas, donde se encuentra las especificaciones de cada actividad.

Tabla 3. 3: Cronograma de planificación de pruebas [Elaboración propia].

Actividades	Días	Responsable	Participantes
Elaborar la estrategia de prueba	Día 1	Coordinador de prueba	Coordinador de prueba
Diseñar la prueba	Día 2	Coordinador de prueba	Coordinador de prueba
Realizar el montaje del entorno de prueba	Día 3	Coordinador de prueba	Coordinador de prueba
Primera iteración de las pruebas	Día 4	Coordinador de prueba	Probador
Corrección de defectos en los artefactos en prueba y actualización de los CP y artefactos de apoyo	Día 5	Jefe de equipo	Desarrollador
Segunda iteración de las pruebas	Día 6	Coordinador de prueba	Probador
Evaluación de los resultados de las pruebas	Día 7	Coordinador de prueba	Probador

3.3.2 Diseño de los escenarios para las pruebas

Con el objetivo de mejorar el margen de confianza, de que se encontrarán todos los defectos, se emplearon los CP para verificar las diversas funcionalidades de un producto, descritas en el formato de las HU.

Para el DCP se utilizarán el diccionario de datos que se muestra en el Anexo E Diccionario de DatosE y las HU.

En la tabla 3.4 que se muestra a continuación se relacionan los CP con las HU especificadas en el capítulo anterior:

Tabla 3. 4: Relación de Casos de Pruebas con HU [Elaboración propia]

Nombre del CP	Historia de Usuario asociada
Autenticar usuario	Autenticar Usuario
Gestionar Caso de Prueba Plantilla	Guardar Caso de Prueba como plantilla
	Modificar Caso de Prueba plantilla

En el DCP se describieron las variables pertenecientes a cada uno y los valores que las mismas deben tomar para la realización de las pruebas. En la figura 3.2 se observa el DCP Autenticar usuario.

Descripción general					
RF2 El sistema permite autenticar usuario					
Condiciones de ejecución					
El usuario debe estar registrado en el sistema con anterioridad.					
SC RF1_Autenticar usuario					
Escenario	Descripción	Nombre de usuario	Contraseña	Respuesta del sistema	Flujo central
EC 1.1 Autenticar usuario de forma correcta	El sistema autentica un usuario de forma correcta.	V mmartell	V Mary.1998	El sistema autentica al usuario y le permite el acceso a las funcionalidades del sistema, según su rol.	1-El usuario accede a la url del portal web y agrega la sentencia "autenticar-usuario" al final de la misma. 2- El sistema muestra una interfaz con el formulario de autenticación. 3- El usuario introduce la información y presiona el botón: "Iniciar sesión".
EC 1.2 Autenticar usuario de forma incorrecta	El sistema no autentica un usuario de forma incorrecta.	V mmartell	I Mary.199	El sistema no autentica al usuario y el borde del campo cambia a color rojo, indicando que es incorrecto.	
		I -usuario	V Mary.1998		
		I mmarte	I Mary.199		
		I mmarte	I Mary.199		
EC 1.3 Autenticar usuario dejando campos vacíos.	El sistema no autentica un usuario dejando campos obligatorios vacíos.	V mmartell	I	El sistema no autentica al usuario y los bordes de los campos cambian a color rojo indicando que son obligatorios.	
		I	V		
		I	Mary.1998		
		I	I		

[Las celdas de la tabla contienen V, I, N/A. V indica válido, I indica inválido y N/A indica que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.]

Figura 3. 2: Diseño del CP Autenticar usuario [Elaboración propia]

3.4 Desarrollo de pruebas de software

Para la calidad de la herramienta se realizarán pruebas a nivel de sistema, evaluando el esfuerzo empleado durante la elaboración de los DCP antes y después de usar la herramienta.

El desarrollo de las pruebas se llevó a cabo en 2 iteraciones:

Iteración 1: se realizaron a un 90% de la aplicación.

Iteración 2: se realizaron al 10 % restante y se realizaron las pruebas no funcionales.

3.4.1 Pruebas funcionales

Las pruebas funcionales son aquellas que se aplican al producto final, y permiten detectar en qué puntos el producto no cumple sus especificaciones, es decir, comprobar su funcionalidad (Palacio, 2009). Con el objetivo de encontrar los errores existentes se planificaron 2 iteraciones.

- En la primera iteración se realizaron:

Pruebas Unitarias basadas en componentes.

Las pruebas unitarias lo que busca es corroborar el comportamiento correcto de una unidad de código. Ayudan a detectar errores que no contemplamos en nuestra fase de desarrollo. Son

un respaldo para futuros cambios en la lógica de nuestros métodos o funciones (Rodríguez Patiño, 2020).

Pruebas de Sistema – Funcionales

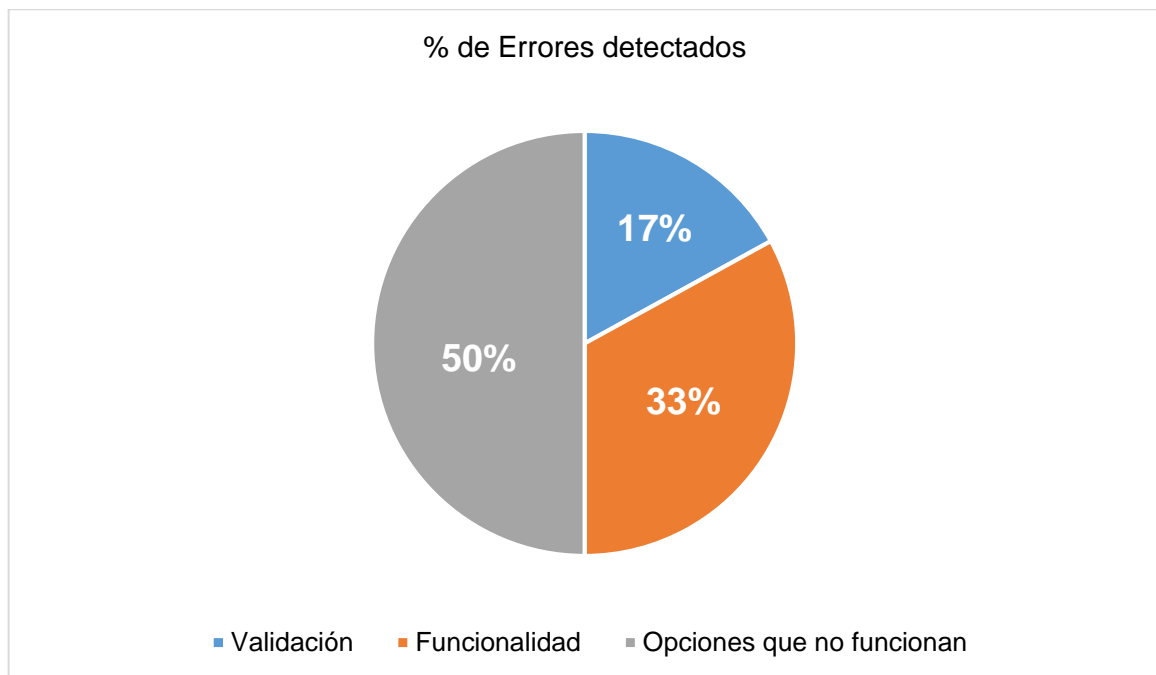
Las pruebas de sistema buscan diferencias entre la especificación funcional del sistema y su comportamiento real. Se debe especificar la operación del software como un todo y aplican la técnica de caja negra. La prueba funcional permite determinar, si el sistema a probar cumple con los requisitos funcionales especificados por el cliente (Ch Ga, 2017).

Luego de realizar estas pruebas se detectaron 9 errores, de ellos se solucionaron 3, estos errores se clasifican en:

1 de validación

2 de funcionalidad

6 opciones que no funcionan



- En la segunda iteración se solucionaron los 6 errores que quedaban pendientes de la 1era iteración.

Pruebas de regresión

En el Anexo F Corrección luego de ejecutadas las pruebas de regresión se muestran algunas imágenes con cambios realizados luego de ejecutar las pruebas de regresión, los errores encontrados fueron 8, se debieron a incompatibilidad entre el *front-end* y el *back-end* debido a nombres de variables. Se corrigieron todos los errores.

Pruebas de integración

Una vez realizadas las pruebas de integración no se detectaron errores en el sistema, ya que todos los componentes y módulos realizaron las acciones para lo que fueron diseñado.

3.4.3 Pruebas no funcionales

Pruebas de Usabilidad

Se hace uso de la “Lista de Chequeo de Usabilidad para sitios web” desarrollada por los especialistas del grupo de Seguridad del Departamento de Evaluación de Productos de Software (DEPSW), perteneciente al Centro Nacional de Calidad de Software (CALISOFT). En la tabla 3.5 se muestran los resultados de estas pruebas:

Tabla 3. 5: Resultado de la lista de chequeo [Elaboración propia].

Categoría de los indicadores	Indicadores	Proceden	Correctos	Incorrectos
Visibilidad del sistema	17	15	15	0
Lenguaje común entre sistema y usuario	11	6	5	1
Libertad y control por parte del usuario	29	20	20	0
Consistencia y estándares	33	30	30	0
Estética y diseño minimalista	18	15	14	1
Prevención de errores	8	5	5	0
Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de los errores	11	7	7	0
Ayuda y documentación	11	8	7	1
Flexibilidad y eficiencia de uso	6	5	5	0
Total	144	111	108	3

Del total de parámetros originales, proceden solo 111, se evaluaron como correctos 108 parámetros, identificando 3 no conformidades, las cuales fueron resueltas para un 100% de usabilidad.

Pruebas de Rendimiento

Participaron en estas pruebas 5 usuarios para comprobar el esfuerzo durante el diseño de 20 escenarios en un proyecto de la UCI, sin el uso de la herramienta y con el uso de la misma como se muestra en la tabla 3.6 se muestra el esfuerzo dedicado al diseño de un escenario sin la herramienta y haciendo uso de ella.

Tabla 3. 6: Esfuerzo dedicado al diseño de escenarios [Elaboración propia]

Proyecto	Total de Escenarios	Sin el uso de la herramienta			Con el uso de la herramienta		
		Total de especialistas	Cant. de escenarios	Cálculo de esfuerzo (horas/hombre)	Total de especialistas	Cant. de escenarios	Cálculo de esfuerzo (horas/hombre)
Portal Web	20	5	4	5h/h	5	5	4h/h

Luego de analizar el esfuerzo empleado para diseñar un escenario, se pudo demostrar que, con el uso de la herramienta, es menor ese esfuerzo durante este proceso.

3.4.4 Triangulación metodológica

La triangulación es considerada como un procedimiento que consiste en recoger y analizar datos, desde distintos ángulos, a fin de contrastarlos e interpretarlos (Avila, González y Licea, 2019). El método permite contrastar los resultados obtenidos para determinar las coincidencias obtenidos en la cuantificación de variables mediante un método cuantitativo, las tendencias y dimensiones que surgen de la aplicación de métodos cualitativos. es la aplicación y combinación de varias metodologías de la investigación en el estudio de un mismo fenómeno (Marín Díaz, 2018).

Una vez sean aplicados los métodos descritos anteriormente se podrá determinar si se cumplió el objetivo de la presente investigación: desarrollar una herramienta para disminuir el esfuerzo empleado en el diseño de casos de pruebas funcionales empleando técnicas de reutilización.

3.5 Conclusiones del capítulo

Luego de llevar a cabo la implementación de la propuesta de solución y elaborada la estrategia de pruebas para la misma se puede concluir que:

- Se definió la tipología de hardware donde se encuentra y ejecuta el sistema desarrollado.
- Se definió la estrategia de prueba a realizar para comprobar el correcto funcionamiento y calidad del sistema.

CONCLUSIONES

Para concluir este Trabajo de Diploma se desarrolló una Herramienta para la generación de casos de prueba funcionales aplicando técnicas de reutilización, para ello:

- El análisis de los principales conceptos asociados a la investigación fue de vital importancia para determinar el alcance de la aplicación.
- El estudio de herramientas homólogas permitió identificar funcionalidades que no deben faltar en la herramienta.
- El estudio de metodologías y tecnologías ayudó a decidir cuál eran las mejores opciones para obtener un sistema con la calidad requerida.
- Para el diseño e implementación de la propuesta fue necesario vincular distintas áreas del conocimiento como lo son ingeniería y programación.
- Con el diseño de la estrategia de pruebas se determinó el cronograma a seguir para realizar las pruebas y cuáles se ejecutarán una vez concluido el proceso de implementación.

RECOMENDACIONES

Una vez concluida la investigación y aprovechando la experiencia obtenida, se proponen las siguientes recomendaciones:

- Diseñar una base de conocimientos para recomendar escenarios de prueba.

REFERENCIAS BIBLIOGRÁFICAS

ALBO CASTRO, M.M. y COCA BERGOLLA, Y., 2020. Importancia de la calidad de la distribución GNU/Linux Nova para la informatización del sistema de salud de Cuba. *Revista Cubana de Información en Ciencias de la Salud* [en línea], vol. 31, no. 4. ISSN 2307-2113. Disponible en: http://scielo.sld.cu/scielo.php?script=sci_abstract&pid=S2307-21132020000400008&lng=es&nrm=iso&tlng=es.

ALTUBE VERA, R., 2021. Qué es Laravel: Características y ventajas. *OpenWebinars.net* [en línea]. Disponible en: <https://openwebinars.net/blog/que-es-laravel-caracteristicas-y-ventajas/>.

ANAYA DE PAEZ, R., 2012. Un acercamiento a la reutilización en ingeniería de software | Revista Universidad EAFIT. [en línea], Disponible en: <https://publicaciones.eafit.edu.co/index.php/revista-universidad-eafit/article/view/1074>.

ANAYA, E., 2020. ¿Cómo desarrollar casos de pruebas para software? - INMEDIATUM. [en línea]. Disponible en: <https://inmediatum.com/blog/ingenieria/como-desarrollar-casos-de-pruebas-para-software/>.

AVILA, H.F., GONZÁLEZ, M.M. y LICEA, S.M., 2019. La triangulación metodológica como método de la investigación científica: Apuntes para una conceptualización. *Didasc@Iia: Didáctica y Educación*, vol. 10, no. 4, pp. 137-146. ISSN 2224-2643.

BALSAMIQ STUDIOS, 2021. Balsamiq. Rapid, Effective and Fun Wireframing Software | Balsamiq. [en línea]. Disponible en: <https://balsamiq.com/>.

CAICEDO SALAZAR, J.A., GUERRERO ARELLANO, H.E. y POMBAR VALLEJOS, P.G., 2017. Sistema de información web transaccional de control de turnos, asistencia y solicitudes de novedades de personal. *Dominio de las Ciencias*, vol. 3, no. 2, pp. 539-580. ISSN 2477-8818.

CAMBARIERI, M., VIVAS, L. y GARCÍA MARTÍNEZ, N., 2019. Marco de trabajo para la aplicación de línea de producto en el desarrollo de software para gobierno digital. *XXV Congreso Argentino de Ciencias de la Computación (CACIC) (Universidad Nacional de Río Cuarto, Córdoba, 14 al 18 de octubre de 2019)* [en línea]. S.l.: s.n., ISBN 978-987-688-377-1. Disponible en: <http://sedici.unlp.edu.ar/handle/10915/91385>.

- CASTAÑEDA ROJAS, E.B., 2016. Propuesta de patrón de diseño de software orientado a prevenir la extracción automatizada de contenido web. En: Accepted: 2016-11-26T17:22:22Z, *Pontificia Universidad Católica del Perú* [en línea], Disponible en: <https://tesis.pucp.edu.pe/repositorio/handle/20.500.12404/7513>.
- CASTILLO, A., 2019. ¿Qué es PHP y cómo funciona? [en línea]. Disponible en: <https://echemosunbitstazo.es/blog/que-es-php-y-como-funciona>.
- CH GA, F., 2017. Pruebas de Sistema. *Mundo Testing* [en línea]. Disponible en: <https://mundotesting.com/pruebas-de-sistema/>.
- COELHO, F.F., 2019. Introducción a Selenium: Cómo funciona, Características y Opciones. *DIGITAL55* [en línea]. Disponible en: <https://www.digital55.com/desarrollo-tecnologia/herramientas-testing-introduccion-selenium/>.
- COLORADO RIVERA, L.P., 2020. Automatización de pruebas funcionales, un complemento para la calidad del software. [en línea]. Disponible en: <https://repository.unimilitar.edu.co/handle/10654/37769>.
- DCS, [sin fecha]. *Plantilla del Plan de pruebas del LPS UCI v5.0*. S.l.: s.n.
- DE SOUZA, I., 2019. ¿Qué es un servidor web y cuáles son sus características? *Rock Content - ES* [en línea]. Disponible en: <https://rockcontent.com/es/blog/que-es-un-servidor/>.
- DEFINICIÓN.DE, 2019. Definición de lenguaje HTML — Definicion.de. *Definición.de* [en línea]. Disponible en: <https://definicion.de/lenguaje-html/>.
- DELGADO GONZÁLEZ, A.L., 2016. *Propuestas para la reutilización en el desarrollo de interfaces de usuario basado en modelos* [en línea]. <http://purl.org/dc/dcmitype/Text>. S.l.: Universidad de Sevilla. Disponible en: <https://dialnet.unirioja.es/servlet/tesis?codigo=46930>.
- DEVS, Q., 2019. ¿Qué es Angular y para qué sirve? *Quality Devs* [en línea]. Disponible en: <https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/>.
- DOMINICODE, 2019. ¿Qué es Typescript? *DOMINICODE.COM* [en línea]. Disponible en: <https://dominicode.com/leccion/que-es-typescript/>.

- EDRAWSOFT, 2021. Qué es el Diagrama de Entidad Relación (ERD). [en línea]. Disponible en: <https://www.edrawsoft.com/es/er-diagram/>.
- FLOOD, C.A., 2017. *Unified Software Engineering Reuse: A Methodology for Effective Software Reuse* [en línea]. Master of Science. S.I.: San Jose State University. Disponible en: https://scholarworks.sjsu.edu/etd_theses/4796.
- GAITÁN PEÑA, C.A., 2017. Líneas de Productos Software: Generando Código a Partir de Modelos y Patrones. *Scientia et Technica*, vol. 22, no. 2, pp. 175-182. ISSN 2344-7214. DOI 10.22517/23447214.9131.
- GALIANA, P., 2021. 26 herramientas de prototipado y usabilidad web. *Thinking for Innovation* [en línea], Disponible en: <https://www.iebschool.com/blog/herramientas-prototipado-analitica-usabilidad/>.
- GROUP, P.G.D., 2021. PostgreSQL. *PostgreSQL* [en línea]. Disponible en: <https://www.postgresql.org/>.
- HERNÁNDEZ, J., 2019. What is Apache? In-Depth Overview of Apache Web Server. *Sumo Logic* [en línea]. Disponible en: <https://www.sumologic.com/blog/apache-web-server-introduction/>.
- HERRERA, C., 2020. Qué es MySQL y para qué sirve - Bloguero Pro. [en línea]. Disponible en: <https://bloguero.pro.com/blog/que-es-mysql-y-para-que-sirve>.
- IBM CORPORATION, 2021. Overview of Rational Quality Manager. [en línea]. Disponible en: <https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/elm/6.0.2?topic=overview-rational-quality-manager>.
- IONOS ESPAÑA S.L.U., 2018. UML, lenguaje de modelado gráfico. *IONOS Digitalguide* [en línea]. Disponible en: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/uml-lenguaje-unificado-de-modelado-orientado-a-objetos/>.
- ISO/IEC 25010, 2011. ISO/IEC 25010 Software Product Quality Requirements and Evaluation (SquaRE). ISO [en línea]. Disponible en:

<https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/57/35733.html>.

ISOTOOLS, 2018. ¿Qué es un checklist y como se debe utilizar? [en línea]. Disponible en: <https://www.isotools.org/2018/03/08/que-es-un-checklist-y-como-se-debe-utilizar/>.

ISTQB, 2018. ISTQB CTFL Syllabus 2018 V3.1. [en línea]. Disponible en: <https://www.istqb.org/downloads/category/2-foundation-level-documents.html>.

JETBRAINS S.R.O, 2021. PhpStorm: The Lightning-Smart IDE for PHP Programming by JetBrains. *JetBrains* [en línea]. Disponible en: <https://www.jetbrains.com/phpstorm/>.

JUAREZ, R., 2020. Pruebas de Software: Historia y Evolución. [en línea]. Disponible en: <https://www.fyccorp.com/articulo-pruebas-de-software:-historia-y-evolucion>.

KUMAR SINGH, S. y SINGH, A., 2016. *SOFTWARE TESTING*. S.I.: Vandana Publications. ISBN 978-81-941110-6-1.

LARMAN, C., 2003. Una introduccion al analisis y diseno orientado a objetos y al proceso unificado. [en línea], Disponible en: https://www.academia.edu/17173013/Una_introduccion_al_analisis_y_diseno_orientado_a_objetos_y_al_proceso_unificado.

LEAVE A COMMENT, 2019. TestLink - Gestor web de pruebas y requisitos gratuito. *Tester House* [en línea]. Disponible en: <https://testerhouse.com/herramienta-testing/testlink/>.

LEE, G., 2020. Tipos de pruebas de software: diferencias y ejemplos. *LoadView* [en línea]. Disponible en: <https://www.loadview-testing.com/es/blog/tipos-de-pruebas-de-software-diferencias-y-ejemplos/>.

LÓPEZ MENDOZA, M., 2020. Qué es un lenguaje de programación. *OpenWebinars.net* [en línea]. Disponible en: <https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion/>.

MARÍN DÍAZ, A., DÍAZ, A.M., CASAÑOLA, Y.T. y HIDALGO, D.B., 2018. Marco de Trabajo para gestionar actividades de calidad. *Revista Cubana de Ciencias Informáticas*, vol. 12, no. 2, pp. 74-88. ISSN 1994-1536.

- MARÍN, R., 2019. Los gestores de bases de datos (SGBD) más usados. *Canal Informática y TICS* [en línea]. Disponible en: <https://revistadigital.inesem.es/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/>.
- MARYQIU1987, 2021. Database normalization description - Office. [en línea]. Disponible en: <https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description>.
- MENZINSKY, A., LÓPEZ, G., PALACIO, J., SOBRINO, M.A., ÁLVAREZ, R. y RIVAS, V., 2020a. Historias de Usuario. , pp. 64.
- MENZINSKY, A., LÓPEZ, G., PALACIO, J., SOBRINO, M.A., ÁLVAREZ, R. y RIVAS, V., 2020b. Historias de Usuario. , pp. 64.
- MONTERO CÁCERES, A.M., 2017. *Mantenimiento de una familia de productos en el dominio de administración de emergencias* [en línea]. Thesis. S.I.: Universidad Católica de la Santísima Concepción. Disponible en: <http://repositoriodigital.ucsc.cl/handle/25022009/1214>.
- MONTESA RAUSELL, P.M., 2017. Development of a knowledge base for knowledge reuse in desing projects. Mini's Interior desing as a case of study. [en línea]. S.I.: Disponible en: https://riunet.upv.es/bitstream/handle/10251/81053/29216867J_TFM_14918966121231395343232295959485.pdf?sequence=2&isAllowed=y.
- MOZILLA, 2020. Learn to style HTML using CSS - Learn web development | MDN. [en línea]. Disponible en: <https://developer.mozilla.org/es/docs/Learn/CSS>.
- MÜLLER, T. y FRIEDENBERG, D., 2011. Probador Certificado. Programa de estudio de nivel básico. [en línea]. S.I.: Disponible en: http://www.sstqb.es/ficheros/sstqb_file97-fb1c34.pdf.
- ORDOÑEZ GUZMÁN, Y.A. y GIRALDO MUÑOZ, E.A., 2016. Método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software : Un estudio de caso en una pequeña organización de software. En: Accepted: 2019-11-29T14:42:31Z [en línea], Disponible en: <http://repositorio.unicauca.edu.co:8080/xmlui/handle/123456789/1713>.

- ORO, L., ALVARADO, Y. y RAMÍREZ PÉREZ, J., 2019. Proceso de diseño del software para un modelo de la calidad en Cuba. *I+D Tecnológico*, vol. 15, pp. 87-98. DOI 10.33412/idt.v15.1.2103.
- ORTEGA LOAIZA, R.V., 2017. ing del software. [en línea]. S.I. Disponible en: <https://es.slideshare.net/RosaOrtega6/ing-del-software-77145198>.
- ORTEGO DELGADO, D., 2017. Los 5 mejores editores PHP. *OpenWebinars.net* [en línea]. Disponible en: <https://openwebinars.net/blog/los-5-mejores-editores-php/>.
- ORTIZ HERRERA, M., 2018. MÉTODOS Y TÉCNICAS PARA LA GESTIÓN DE PROYECTOS SOFTWARE. [en línea]. Disponible en: <http://bibing.us.es/proyectos/abreproy/70193>.
- OSCCO HUANGAL, C.J. y MONTOYA MALDONADO, L., 2016. Modelamiento software. [en línea]. S.I. Disponible en: <https://es.slideshare.net/CristhianJOsccoHuang/modelamiento-software>.
- PALACIO, L.G., 2009. Método para generar casos de prueba funcional en el desarrollo de software. *Revista Ingenierías Universidad de Medellín*, vol. 8, no. 15 Sup. 1, pp. 29-36. ISSN 2248-4094.
- PARRA VALVERDE, E., 2020. Estudio del control de calidad en procesos de desarrollo de software. Aplicación práctica de pruebas automatizadas a un sitio web de comercio electrónico. En: Accepted: 2020-08-28T11:16:51Z [en línea], Disponible en: <https://uvadoc.uva.es/handle/10324/42018>.
- PEIRÓ, R., 2020. Base de datos - Qué es, definición y concepto | 2021 | Economipedia. [en línea]. [Consulta: 5 octubre 2021]. Disponible en: <https://economipedia.com/definiciones/base-de-datos.html>.
- PÉREZ, A., 2016. ¿Qué son las metodologías de desarrollo de software? *OBS Business School* [en línea]. Disponible en: <https://www.obsbusiness.school/blog/que-son-las-metodologias-de-desarrollo-de-software>.
- POLO USAOLA, M., 2013. Desarrollo de software basado en reutilización. En: Accepted: 2021-02-16T12:18:42Z [en línea], Disponible en: <https://repositorio.inci.gov.co/handle/inci/434>.

- PUENTE CEDILLO, O.M., 2013. Material teórico completo para la materia de Programación Web. [en línea]. Disponible en: <https://programacionwebisc.wordpress.com/2-3-lenguajes-de-programacion-del-lado-del-servidor/>.
- RAMÍREZ, P., 2017. Modelo Vista Controlador MVC: ¿Qué es y para qué sirve? - ITSoftware. *ITSoftware | Apps | Software | Data Analytics* [en línea]. Disponible en: <https://itsoftware.com.co/content/modelo-vista-controlador-mvc-sirve/>.
- RAMOS, A., 2015. Redmine, software libre de gestión de proyectos. [en línea]. Disponible en: <https://www.incubaweb.com/redmine-software-libre-de-gestion-de-proyectos/>.
- RED HAT, INC, 2019. What is an IDE? [en línea]. Disponible en: <https://www.redhat.com/en/topics/middleware/what-is-ide>.
- RG, A., 2017. ¿Qué es un framework? |. [en línea]. Disponible en: <http://www.araceliroman.com/blog/que-es-un-framework/>.
- RODRÍGUEZ BARAJAS, C.T., 2018. Impacto de los requerimientos en la calidad de software | Tecnología Investigación y Academia. [en línea], Disponible en: <https://revistas.udistrital.edu.co/index.php/tia/article/view/7607>.
- RODRÍGUEZ PATIÑO, E., 2020. Que son la spruebas unitarias e importancia de estas. [en línea]. Disponible en: <https://anexsoft.com/que-son-las-pruebas-unitarias-e-importancia-de-estas>.
- RUEDA PATIÑO, A.A., CRUZ MOSQUERA, H.F. y LONDOÑO ROJAS, J.A., 2016. *Propuesta de Automatización de Casos de Prueba Para Aseguramiento de Calidad en el Desarrollo de Software*. [en línea]. Thesis. S.I.: Posgrado. Disponible en: <https://repository.uniminuto.edu/handle/10656/5647>.
- SÁNCHEZ, T.R., 2015. *Metodología de desarrollo para la Actividad productiva de la UCI* [en línea]. 2015. S.I.: s.n. Disponible en: <http://mejoras.prod.uci.cu/>.
- SANTOS ALONSO, J.M., 2020. La soberanía tecnológica de la gestión de las tecnologías de la información y la Política Tecnológica. *Serie Científica de la Universidad de las Ciencias Informáticas*, vol. 13, no. 6, pp. 15-24. ISSN 2306-2495.

- SARMIENTO, M., 2017. Normalización de base de datos - marcosarmiento.com. [en línea]. Disponible en: <http://www.marcossarmiento.com/2017/06/28/normalizacion-de-base-de-datos/>.
- SOMMERVILLE, I., 2005. Ingeniería de Software - Ian Sommerville 7a Edición. [en línea], Disponible en: https://www.academia.edu/15059886/Ingenieria_de_Software_Ian_Sommerville_7a_Edicion.
- TORO LAZO, A. y GÁLVEZ BOTERO, J.G., 2016. Especificación de requisitos de software: una mirada desde la revisión teórica de antecedentes. *Entre Ciencia e Ingeniería*, vol. 10, no. 19, pp. 108-115. ISSN 2539-4169.
- VARGAS FANDIÑO, J.C., SANDOVAL RAMÍREZ, J.J. y VERA RIVERA, F., 2020. Implementación de un repositorio para el catálogo, búsqueda y uso de componentes software reutilizables en el desarrollo de aplicaciones web. *Revista UIS Ingenierías*, vol. 19, no. 2, pp. 11-20. ISSN 2145-8456. DOI 10.18273/revuin.v19n2-2020002.
- VÁZQUEZ-INGELMO, A. y THERÓN, R., 2020. Beneficios de la aplicación del paradigma de líneas de productos software para generar dashboards en contextos educativos. *RIED. Revista Iberoamericana de Educación a Distancia*, vol. 23, no. 2, pp. 169. ISSN 1390-3306, 1138-2783. DOI 10.5944/ried.23.2.26389.
- VEGA LLOBELL, A.T., 2018. Pruebas funcionales automatizadas para aplicaciones Web: usando Selenium para aplicar pruebas de regresión automatizadas. [en línea]. Disponible en: <https://riunet.upv.es/handle/10251/111170>.
- VELÁZQUEZ CINTRA, A., FEBLE ESTRADA, A. y MERCED LEN, S., 2018. Evaluación de productos de software desde la perspectiva del cliente. [en línea]. S.I.: Disponible en: <http://www.informaticahabana.cu/es/node/3711>.
- VELÁZQUEZ GODOY, Y., VELÁZQUEZ CINTRA, A. y COLLADO ROLO, L., 2020. Diseño de herramienta para casos de pruebas funcionales en la universidad de las ciencias informáticas. *Serie Científica de la Universidad de las Ciencias Informáticas*, vol. 13, no. 1, pp. 47-61. ISSN 2306-2495.

VISUAL PARADIGM, 2021. Visual Paradigm Frequently Asked Questions. [en línea]. Disponible en: <https://www.visual-paradigm.com/support/faq.jsp>.

ANEXOS

A Cuestionario para obtener información sobre la reutilización de elementos durante el DCP funcionales.

Con este cuestionario se pretende identificar los elementos que pueden ser reutilizados durante el diseño de casos de pruebas funcionales.

Se le asegura confidencialidad y anonimato a su respuesta

Organización/Área a la que pertenece: _____

Rol: _____

Grado científico: _____

Años de experiencia en diseños de casos de prueba: _____

1. Indique en cuántos procesos de diseño de casos de prueba funcionales ha participado:

___ Menos de 5 ___ De 5 a 10 ___ Más de 10

2. ¿Considera usted que el diseño incorrecto de los casos de prueba influye en la calidad del producto?

___Sí ___No

3. ¿Durante el proceso de diseño de casos de pruebas ha identificado usted elementos que puedan ser reutilizados en dos o más productos?

___Sí ___No

a) En caso de ser una respuesta positiva, diga cuáles: _____

4. En una escala ascendente del 1 al 10, marcando con un (x), la importancia que le otorga a la reutilización de elementos en el diseño de casos de pruebas funcionales.

1	2	3	4	5	6	7	8	9	10

5. Considera usted que el esfuerzo que emplea en el diseño de casos de pruebas funcionales es:

___Muy poco ___Poco ___Adecuado ___Mucho ___ Demasiado

6. Teniendo en cuenta los elementos que forman un caso de prueba, señale en cuál usted cree podría reutilizarse en proyectos de naturaleza similar:

Variables

Nombre del Caso de Prueba

Nombre del producto

Partición

Otras

a) En caso de que haya marcados "Otras" especifique cuáles:

7. Según su experiencia, ¿considera oportuno contar con una herramienta que le permita crear plantillas genéricas para facilitar el diseño de casos de pruebas?

Sí No

a) Si su respuesta fue positiva, crearía usted plantillas asociadas a:

escenarios tipo para algunas funcionalidades

casos de prueba tipo

producto tipo

otros (cuáles _____)

8. ¿Utiliza usted alguna herramienta que permita gestionar casos de prueba?

Sí No

9. ¿Conoce usted herramientas que permitan reutilizar variables a la hora de crear CP?

Sí__ No__

a) Si su respuesta fue positiva indique cuáles: _____

b) ¿Qué opinión usted tiene sobre la importancia de la reutilización de elementos que conforman el diseño de casos de pruebas funcionales?

B Guía de entrevista para obtener información sobre la reutilización de elementos durante el DPC funcionales como actividad del proceso de pruebas

Nombre: _____ **Especialidad:** _____

¿Qué le gusta más de esa profesión? _____

¿Ha realizado estudios en esas temáticas? ¿En cuáles? _____

¿Dónde trabaja ahora? _____ ¿Qué cargo ocupa? _____

¿Cuáles considera son sus habilidades en esa área? _____

¿Cuánto tiempo lleva ejerciendo esa responsabilidad? _____

¿Le gusta dirigir un equipo? ¿Por qué? _____

¿Cuál es el mayor reto de su equipo de trabajo en la actualidad? _____

1. ¿Considera necesario reutilizar elementos en el proceso de diseño de casos de pruebas funcionales? ¿Por qué?
2. ¿Considera que reutilizando elementos disminuiría el tiempo empleado en diseño de casos de prueba?
3. ¿Cuáles son los elementos más comunes en el diseño de casos de pruebas funcionales?
4. ¿Durante el proceso de diseño de casos de pruebas ha identificado usted elementos que puedan ser reutilizados en dos o más productos? ¿Cuáles?
5. ¿Actualmente reutilizan elementos en el diseño de casos de prueba? ¿Cuáles?
6. Si reutilizan elementos, ¿Cómo lo hacen?
7. ¿Considera oportuno contar con una herramienta que le permita crear plantillas genéricas para facilitar el diseño de casos de pruebas?

C Principios básicos que guían las pruebas del software

Principio 1: El proceso de prueba demuestra la presencia de defectos.

- El proceso de prueba puede probar la presencia de defectos.
- El proceso de prueba no puede demostrar la ausencia de defectos.

Principio 2: No es posible realizar pruebas exhaustivas.

- Pruebas exhaustivas (".exhaustive testing").
- Explosión de casos de prueba ("test case explosion").
- Prueba de Muestra ("sample test").

Principio 3: Pruebas tempranas (".early testing").

- Cuanto más temprana es la detección de un defecto, menos costosa es su corrección.
- La preparación de una prueba también consume tiempo.

Principio 4: Agrupamiento de defectos ("defect clustering").

- Encuentre un defecto y encontrará más defectos cerca".
- Los probadores ("testers") deben ser flexibles.

Principio 5: Paradoja del pesticida.

- Repetir pruebas en las mismas condiciones no es efectivo.
- Las pruebas deben ser revisadas/modificadas regularmente para los distintos módulos de código.

Principio 6: Las pruebas dependen del contexto.

- Las pruebas se llevan a cabo de forma diferente en diferentes contextos.
- Objetos de prueba diferentes son probados de forma diferente.
- Entorno de prueba vs. entorno de producción.

Principio 7: La falacia de la ausencia de errores.

- Un proceso de prueba adecuado detectará los fallos más importantes.
- En la mayoría de los casos el proceso de prueba no detectará todos los defectos del sistema, pero los defectos más importantes deberían ser detectados.
- Esto por sí solo no prueba la calidad del software.

D Requisitos correspondientes a cada iteración

Requisito	Iteración	Requisito	Iteración
Módulo de administración		Módulo de Diseño de Casos de Prueba Funcionales	
Registrar usuario	1	Crear particiones de un campo	1 y 2
<i>Autenticar usuario</i>	1	Modificar particiones de un campo	2
Cerrar sesión	1	Listar particiones de un campo	2
Listar trazas	1	Eliminar particiones de un campo	2
Crear permisos	1	Crear casos de prueba	1 y 2
Modificar permisos	1	Cargar casos de pruebas	2
Listar permisos	1	Modificar casos de prueba	2
Eliminar permisos	1	Listar casos de prueba	2
Mostrar permisos	1	Mostrar casos de prueba	2
Asignar permisos	1	Eliminar casos de prueba	1 y 2
Quitar permisos	1	Exportar casos de prueba	1 y 2
Crear usuario	1	Cargar casos de pruebas plantillas	2
Modificar usuario	1	Modificar casos de prueba plantilla	2
Listar usuario	1	Listar casos de prueba plantilla	2
Mostrar usuario	1	Mostrar casos de prueba plantilla	2
Eliminar usuario	1	Eliminar casos de prueba plantilla	2
Activar usuario	1	Guardar casos de prueba como plantilla	1 y 2
Desactivar usuario	1	Exportar caso de prueba plantilla	2
Crear rol	1	Crear escenario	1 y 2
<i>Modificar rol</i>	1	Cargar escenario	2
<i>Listar rol</i>	1	Modificar escenario	2
Mostrar rol	1	Listar escenario	2
<i>Eliminar rol</i>	1	Mostrar escenario	2
Asignar rol	1	Eliminar escenario	2
Quitar rol	1	Cargar escenario plantillas	2
<i>Crear proyecto</i>	1	Modificar escenario plantilla	2

<i>Modificar proyecto</i>	1	Listar escenario plantilla	2
<i>Listar proyecto</i>	1	Mostrar escenario plantilla	2
Añadir usuario	1	Eliminar escenario plantilla	2
Mostrar proyecto	1	Guardar escenario como plantilla	1 y 2
<i>Eliminar proyecto</i>	1	Crear variable	1 y 2
Crear producto	1	Cargar variables	2
Modificar producto	1	Modificar variable	2
Listar producto	1	Listar variable	2
Mostrar producto	1	Mostrar variable	2
Eliminar Producto	1	Eliminar variable	2
Crear tipo de producto	1	Crear regla	2
Modificar tipo de producto	1	Listar regla	2
Listar tipo de producto	1	Eliminar regla	2
Eliminar tipo de producto	1		
Crear centro de desarrollo	1		
Modificar centro de desarrollo	1		
Listar centro de desarrollo	1		
Mostrar centro de desarrollo	1		
Eliminar centro de desarrollo	1		
Leyenda: Nuevo Se mantiene <i>Modificado</i>			

E Diccionario de Datos

Nombre de la base de datos	Nombre de la tabla de la base de datos	Variable (atributo)	Descripción	Tipo de dato
herramienta	casos_pruebas y casos_pruebas_plantillas	producto_id	Especifica el producto al cual se le va a realizar el caso de prueba o la plantilla	Número entero
herramienta	casos_pruebas	plantilla_id	Especifica la plantilla que se utiliza	Número entero
herramienta	casos_pruebas y casos_pruebas_plantillas	nombre	Especifica el nombre del caso de prueba o la plantilla	Alfanumérico
herramienta	escenarios	plantilla_id	Especifica la plantilla que se utiliza	Número entero
herramienta	escenarios y escenarios_plantillas	nombre	Especifica el nombre del escenario la plantilla	Alfanumérico
herramienta	escenarios y escenarios_plantillas	descripcion	Describe lo que debe hacer el escenario	Alfanumérico
herramienta	escenarios y escenarios_plantillas	respuesta	Describe cual será la respuesta del sistema	Alfanumérico
herramienta	centros_desarrollo	nombre	Especifica el nombre del centro de desarrollo	Alfanumérico
herramienta	centros_desarrollo	descripcion	Describe brevemente lo que se hace en el centro de desarrollo	Alfanumérico
herramienta	centros_desarrollo	siglas	Especifica las siglas con las que se identifica el centro de desarrollo	Alfanumérico
herramienta	particiones	variable_id	Especifica la variable que se desea particionar	Número entero
herramienta	particiones	representante	Especifica el representante de esa partición	Alfanumérico
herramienta	particiones	limite_inferior	Especifica el límite inferior de la partición	Número con decimales
herramienta	particiones	limite_superior	Especifica el límite superior de la partición	Número con decimales
herramienta	variables	nombre	Especifica el nombre de la variable	Alfanumérico
herramienta	variables	tipo_dato	Especifica el tipo de dato de la variable	Alfanumérico
herramienta	reglas	regla	Especifica la regla que se le crea a la variable	Alfanumérico
herramienta	proyecto	centro_desarrollo_id	Especifica el centro de desarrollo al cual pertenece el proyecto	Número entero
herramienta	proyecto	nombre	Especifica el nombre del proyecto	Alfanumérico
herramienta	producto	proyecto_id	Es el proyecto al cual pertenece el producto	Número entero

herramienta	producto	tipo_producto_id	Especifica el tipo de producto	Número entero
herramienta	producto	nombre	Especifica el nombre del producto	Alfanumérico
herramienta	roles	nombre	Especifica el nombre del rol	Alfanumérico
herramienta	roles	rol	Especifica el rol	Alfanumérico
herramienta	permisos	nombre	Especifica el nombre del permiso	Alfanumérico
herramienta	permisos	permiso	Especifica el permiso	Alfanumérico

F Corrección luego de ejecutadas las pruebas de regresión

```

20     producto: FormGroup = this.formBuilder.group({
21         nombre: ['', [Validators.required]],
-       tipo: ['', [Validators.required]],
-       proyecto: ['', [Validators.required]],
22 +       tipo_producto_id: ['', [Validators.required]],
23 +       proyecto_id: ['', [Validators.required]],
24     });

```

```

25         header: 'No.',
-       column: 'no',
26 +       column: 'id',
27         show: true,
28         type: ColumnType.Regular
29     },
30     {
31         header: 'Producto',
-       column: 'producto',
32 +       column: 'nombre',
33         show: true,
34         type: ColumnType.Regular
35     },
36     {
37         header: 'Tipo de Producto',
-       column: 'tipoProducto',
38 +       column: 'tipo.nombre',
39         show: true,
40         type: ColumnType.Regular
41     },
42     {
43         header: 'Proyecto',
-       column: 'proyecto',
44 +       column: 'proyecto.nombre',
45         show: true,
46         type: ColumnType.Regular

```