PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Escuela de Posgrado

AUTOMATIZACIÓN DE PRUEBAS DE ACEPTACIÓN EN EL PROCESO DE DESARROLLO DE SOFTWARE

Tesis para obtener el grado académico de Magíster en Informática con mención en Ingeniería de Software que presenta:

Carlos Alberto Pesantes Robles

Asesor:

Mg. Luis Alberto Flores García

Lima, 2023

INFORME DE SIMILITUD

Yo, **Luis Alberto FLORES GARCÍA**, docente de la Escuela de Posgrado de la Pontificia Universidad Católica del Perú, asesor de la tesis titulada **"Automatización de pruebas de aceptación en el proceso de desarrollo de software"**, del autor **Carlos Alberto PESANTES ROBLES**, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de **35%**. Así lo consigna el reporte de similitud emitido por el software Turnitin el **04/10/2023**.
- He revisado con detalle dicho reporte y la tesis y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:

Lima, 06 de octubre del 2023.

<table>
<tr><td colspan="2">Apellidos y nombres del asesor:
Flores García, Luis Alberto</td></tr>
<tr><td>DNI: 10772024</td><td rowspan="2">Firma
  </td></tr>
<tr><td>ORCID: 0000-0002-1359-283X</td></tr>
</table>

RESUMEN

El presente proyecto nace como resultado de la necesidad de las empresas de desarrollo de software de optimizar el proceso de desarrollo de software (desarrollo, testing y puesta en producción del producto de software), mediante el uso de herramientas de vanguardia diseñadas con dicho fin.

En este trabajo se ha diseñado e implementado un sistema web que permite la automatización de las pruebas de aceptación de los analistas a cargo del aseguramiento de la calidad del producto de software. El sistema ha sido desarrollado con la metodología ágil eXtreme Programming, con el objetivo de poner énfasis en la adaptabilidad, es decir, simplificar el diseño, agilizar el desarrollo (desarrollo guiado por comportamiento) y facilitar el mantenimiento.

El principal resultado que se obtuvo mediante el uso del sistema implementado fue mejorar los tiempos en la ejecución de pruebas de aceptación realizadas por los analistas de calidad. También se consiguió mejorar la calidad del proceso de pruebas mediante la reducción de los defectos encontrados en la etapa posterior – Pruebas en UAT. Ambos resultados fueron conseguidos a través de indicadores cuantitativos, los cuales fueron evaluados en proyectos de software de 3 diferentes tamaños (pequeño, mediano, grande).

Al terminar este trabajo, se puede afirmar que el diseño e implementación del sistema web propuesto, reduce el tiempo del ciclo de pruebas de aceptación, así como aumenta la calidad del software entregado al cliente.

Palabras clave: Automatización, Pruebas, Desarrollo de Software, Desarrollo Guiado por Comportamiento.

ABSTRACT

The present project was born because of the need of the software development companies to optimize the process of software development (development, testing and release the software product), using avant-garde tools designed for this purpose.

In this work, a web system has been designed and implemented that allows the automation of the acceptance tests of the analysts in charge of assuring the quality of the software product. The system has been developed with the agile methodology eXtreme Programming, with the aim of emphasizing the adaptability, that is, simplifying the design, speeding the development (behavior driven development) and facilitating the maintenance.

The main result obtained using the implemented system was to improve the times in the execution of acceptance tests carried out by the quality analysts. It was also possible to improve the quality of the testing process by reducing the defects found in the later stage - Tests in UAT. Both results were obtained through quantitative indicators, which were evaluated in software projects of 3 different sizes (small, medium, large).

At the end of this work, it was possible to affirm that the design and implementation of the proposed web system optimizes the cycle of acceptance tests.

Keywords: Automation, Testing, Software Development, Behavior Driven Development

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO I: PRESENTACION DEL PROYECTO

En el presente capítulo se hace una introducción a la investigación que se llevó a cabo, así como sus alcances, limitaciones y los procedimientos realizados para alcanzar los objetivos.

1.1 Introducción

En los últimos años se han experimentado grandes cambios en el área de la computación, tanto en la proliferación de nuevas tecnologías, metodologías y enfoques de desarrollo que han repercutido en las Organizaciones actuales, como a la inversa, cambios en los requerimientos y necesidades a nivel Organizacional han repercutido en la forma de hacer y ejecutar software [1].

Esta necesidad del cambio es en donde nuevos conceptos como **Integración Continua (Continous Integration) [4], Testing Continuo (Continous Testing) y Entrega Continua (Continous Delivery)** comienzan a nacer y toman un lugar importante en el día a día de las empresas de desarrollo de Software. Esto debido a que se reducen los tiempos en las actividades que se realiza, facilita la construcción de software y mejora la calidad del mismo [2].

Es así como empresas que se dedican a la construcción de Software, adoptan dichos conceptos y el logro de los mismos se convierte en un objetivo constante. Sin embargo, la realización de dicho proceso que se inicia con la toma de requerimientos y termina con la entrega del producto terminado, no se encuentra automatizado en varias de las etapas del desarrollo dentro de la empresa.

Este problema origina pérdida de tiempo en el proceso de desarrollo como generar defectos provocados por una ineficiente integración de código o sobrescribir requerimientos previos ya implementados. Así como rehacer tareas en la etapa de Testing del producto, cuando los defectos son encontrados y al tener que probar dichos defectos hasta que la prueba sea exitosa. [2]

Actualmente la empresa donde se realizó el caso de estudio se dedica al desarrollo de software utilizando un desarrollo basado en comportamiento (BDD: Behavior driven Development) y usando metodologías agiles como Scrum y Kanban, cuenta con una

cultura DEVOPS en desarrollo, y se encuentra en la búsqueda de la automatización en la mayoría de los procesos desde la toma de requerimientos hasta la entrega del producto terminado.

La empresa cuenta con un proceso de integración continua, y ha reducido los tiempos en la mayoría de las fases del desarrollo. Sin embargo, realizar las pruebas de aceptación y asegurar la calidad del producto de software se han convertido en cuello de botella, debido a las tareas repetitivas y de alta demanda de tiempo. Como resultado los tiempos estimados para un proyecto aumentan, ocasionando un incremento en el costo del desarrollo para el cliente, impactando negativamente la confianza puesta en la empresa.

1.2 Objetivos

1.2.1 Objetivo General

El presente trabajo tiene como objetivo la implementación de un sistema Web para automatizar las pruebas de aceptación, así como la evaluación de su impacto en una empresa de desarrollo de software.

1.2.2 Objetivos Específicos

A continuación, se presenta la lista de objetivos específicos:

OE-01. Desarrollar e implementar un Sistema Web para automatizar las pruebas de aceptación en el proceso de desarrollo de software guiado por comportamiento.

OE-02. Reducir el tiempo que se emplea en realizar las pruebas de aceptación en nuevos proyectos de software de diferentes tamaños.

OE-03. Mantener o disminuir la cantidad de errores encontrados después de la realización de las pruebas de aceptación (Pruebas en UAT) en nuevos proyectos de software de diferentes tamaños.

OE-04. Definir de un modelo estándar para la realización de las pruebas de aceptación.

1.3 Resultados Esperados

OE-01.RE-1 Un sistema Web que automatice las pruebas de aceptación realizadas por el analista de calidad.

OE-02.RE-1 Comparativo del tiempo empleado en la realización de las pruebas de aceptación antes y después de la implementación del sistema web.

OE-03.RE-1 Comparativo de la cantidad de errores encontrados en la realización de las pruebas de UAT antes y después de la implementación del sistema web.

OE-04.RE-1 Documento sobre el marco de trabajo en la realización de las pruebas de aceptación.

1.4 Justificación

Una de las necesidades más apremiantes de las empresas de desarrollo de software es la de poder reducir los tiempos al máximo en todas las etapas del desarrollo de software automatizando las tareas repetitivas, y reduciendo la intervención humana en la realización de dichas tareas manteniendo o mejorando la calidad [5][6].
De esta manera se logra que las tareas en las pruebas de software sean más sencillas y económicas (reducción de costos).

Es así como la industria del software que adopta metodologías agiles como Scrum o Kanban, busca poder optimizar al máximo el proceso mismo de desarrollo, así como asegurar la calidad de sus productos, utilizando herramientas que le permitan lograr dichos objetivos [6].

Uno de los aportes de la realización de este trabajo incluye la definición de un modelo estándar en la realización de las pruebas de aceptación que pueda ser aplicado a todos los proyectos de desarrollo de sistemas dentro de la compañía en los diferentes clientes.

Es por ello por lo que a través de este estudio y con la implementación de un sistema web se pretende desarrollar una manera efectiva de realizar la ejecución de las pruebas de aceptación en el proceso de desarrollo.

1.5 Hipótesis

La implementación de un Sistema web para la automatización de las pruebas de aceptación reduce los tiempos en el proceso de Desarrollo y asegura la calidad del producto de software resultado del desarrollo en la empresa.

1.6 Límites Del Proyecto

Entre las principales limitaciones identificadas en esta investigación se puede mencionar la siguiente:

- Para el análisis de las actividades realizadas por los analistas de calidad en las pruebas de aceptación, se tomará en cuenta sólo los proyectos a DIRECTV.
- Se tomará en cuenta como período de evaluación los proyectos realizados en el año 2016 en la empresa de desarrollo.
- El modo de autenticación al sistema está limitada a cuentas de dominio en Microsoft Azure [34], por lo que el uso del sistema solo podrá ser realizado por trabajadores de la empresa de desarrollo.

1.7 Métodos Y Procedimientos

A continuación, se detallan los procedimientos realizados en esta investigación:

a. **Análisis del Proceso**

   Definir las tareas realizadas por los analistas de calidad en la realización de las pruebas de aceptación, así como las herramientas usadas en este proceso.

b. **Definición del Producto Software Por Desarrollar**

   El Sistema Web se desarrollará tomando en cuenta las prácticas de la programación extrema o eXtreme Programming (XP) [30][31].

   - **Equipo Completo**, se tomó en cuenta las personas que tienen algo que ver con el proyecto, incluyendo los analistas de calidad y el desarrollador.
   - **Planificación**, se consideró las historias de usuario, un plan de entregas y un plan de iteraciones (descritas en el módulo 4).
   - **Diseño Simple,** se hizo lo mínimo imprescindible de la forma más sencilla, manteniendo un código simple.
   - **Desarrollo guiado por pruebas,** se escribieron las pruebas unitarias previamente a cada desarrollo.

- **Disponibilidad del Cliente,** el cliente de la aplicación son los analistas de calidad, los cuales estuvieron disponibles durante todo el proyecto.
- **Integración Continua,** se mantuvo la integración continua y la ejecución de las pruebas unitarias en Teamcity [14].

c. Análisis Estadístico de Resultados

Se realizaron métodos de procesamiento de información para determinar si el uso del producto reduce el tiempo empleado en la fase de pruebas de aceptación, manteniendo o mejorando la calidad del producto.

Indicadores Cuantitativos:

I1. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (Pequeño: 0 – 80 horas).

I2. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (Mediano: 81 – 120 horas).

I3. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (Grande: 121 +).

I4. Numero de errores Promedio en las pruebas de UAT.
(Pequeño: 0 – 80 horas).

I5. Numero de errores Promedio en las pruebas de UAT.
(Mediano: 81 – 120 horas).

I6. Numero de errores Promedio en las pruebas de UAT.
(Grande: 121 + horas).

CAPÍTULO II: MARCO CONCEPTUAL

En el presente capítulo, se presentan algunos conceptos necesarios para el entendimiento del proyecto.

2.1 Introducción

Se hará una breve descripción sobre el desarrollo guiado por pruebas (TDD), desarrollo guiado por comportamiento (BDD), las metodologías ágiles, integración continua, Entrega Continua, la cultura DEVOPS, y las ventajas de usarlo en el desarrollo de software. Se introducirá conceptos y términos clave para el entendimiento del resto del documento. Se hará un breve resumen sobre las fases que involucra, así como los artefactos utilizados.

2.2 Desarrollo Guiado por Pruebas (TDD)

El Desarrollo guiado por pruebas (TDD) es un proceso de desarrollo de software originado en Extreme Programming (XP) inventado por Kent Beck, que se basa en la repetición de una serie de ciclos de desarrollo cortos y continuos [35].

El Desarrollo guiado por pruebas (TDD) es una forma de programación que fomenta un buen diseño y es un proceso disciplinado que ayuda a evitar errores de programación. TDD define las pruebas al inicio del proceso, que eventualmente crean un sistema de alarma muy efectivo para proteger nuestro código contra la regresión [37]. El flujo completo se muestra en la figura 1.

TDD puede conducir a un código más modularizado, flexible y extensible; La naturaleza temprana y frecuente de las pruebas ayuda a detectar los defectos al principio del ciclo de desarrollo, evitando que se conviertan en problemas endémicos y costosos [35]. Además de esto, se practica completamente:

- "Déjalo simple" ("*Keep it Simple, Stupid*!") conocido como el principio KISS
- "No lo vas a necesitar" ("*You ain't gonna need it*") conocido como el principio YAGNI.

El flujo de trabajo para TDD es el siguiente:

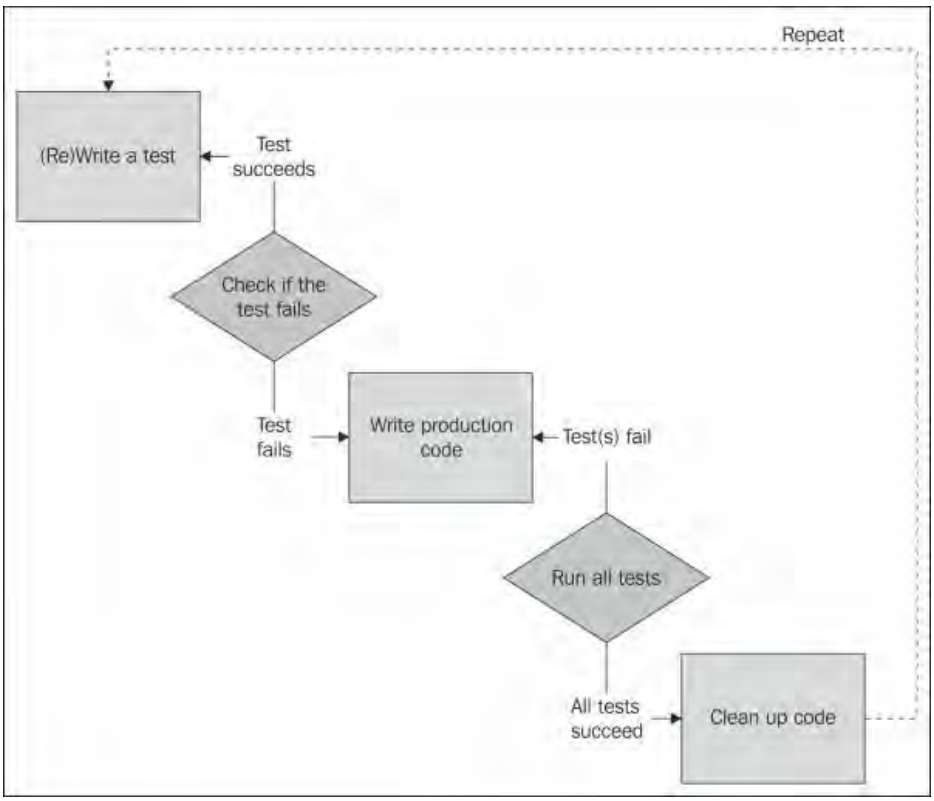


Figura 1: Flujo de Trabajo TDD [35]

2.3 Desarrollo Guiado por Comportamiento (BDD)

BDD se basa en TDD (Test driven Development); hereda todos los beneficios y muchos de los principios / prácticas de TDD, pero avanza un paso: BDD combina TDD con ideas de diseño impulsado por dominio (DDD: Domain Driven Development) y análisis y diseño orientado a objetos que proporcionan equipos de desarrollo de software y gente de negocios herramientas compartidas y un proceso compartido para colaborar en el desarrollo de software [35]. El inventor de BDD, Dan North, lo definió de la siguiente manera:

> "BDD es una metodología ágil de segunda generación, de afuera hacia adentro, de múltiples partes interesadas, a múltiples escalas y alta automatización. Describe un ciclo de interacciones con productos bien definidos, lo que resulta en la entrega de software probado que genera valor" [35].

BDD se enfoca en implementar la característica comercializable mínima (MMF: *Minimum Marketable Feature*) que producirá el mayor valor. El equipo comercial y el equipo de desarrollo pueden cooperar en un lenguaje común; esto reduce significativamente la incomprensión y elimina el desperdicio, el código innecesario y la funcionalidad. [35]

Los profesionales de BDD usan conversaciones sobre ejemplos concretos de comportamiento del sistema para ayudar a comprender cómo las características le darán valor al negocio. BDD alienta a los analistas de negocios, desarrolladores de software y evaluadores a colaborar más estrechamente permitiéndoles expresar los requisitos de una manera más comprobable, de una forma que tanto el equipo de desarrollo como las partes interesadas del negocio puedan comprender fácilmente [35]. Las herramientas de BDD pueden ayudar a convertir estos requisitos en pruebas automatizadas que ayudan a guiar al desarrollador, verificar la función y documentar lo que hace la aplicación. [36]

BDD no es una metodología de desarrollo de software en sí misma. No reemplaza a Scrum, XP, Kanban, RUP ni a la metodología que esté utilizando actualmente. BDD incorpora, desarrolla y mejora las ideas de muchas de estas metodologías [36].

2.4 Cultura DEVOPS

DevOps (*Development plus Operations*) ha tomado recientemente el centro del escenario en el SDLC (*Software Development Life Cycle*). DevOps ofrece marcos de procesos mejorados con herramientas de código abierto para integrar todas las fases del ciclo de vida de la aplicación y garantizar que funcionen como una unidad cohesionada [38]. Ayuda a alinear y automatizar el proceso en las fases de desarrollo, prueba, implementación y soporte. Incluye las mejores prácticas, como repositorios de código, automatización de compilación, implementación continua y otros [38].

Nacido de la necesidad de mejorar la agilidad de prestación del servicio de TI el movimiento DevOps enfatiza la comunicación, colaboración y la integración entre desarrolladores de software y operaciones de TI. En lugar de ver a estos dos grupos como silos que se van pasando las cosas pero que no trabajan realmente juntos, DevOps reconoce la interdependencia del desarrollo del software y las operaciones de TI y ayuda a una organización a producir software y servicios de TI de forma más rápida, con frecuentes interacciones [7].

Como se puede observar en la figura 2, la adopción de DevOps ha experimentado un crecimiento constante año tras año [38].

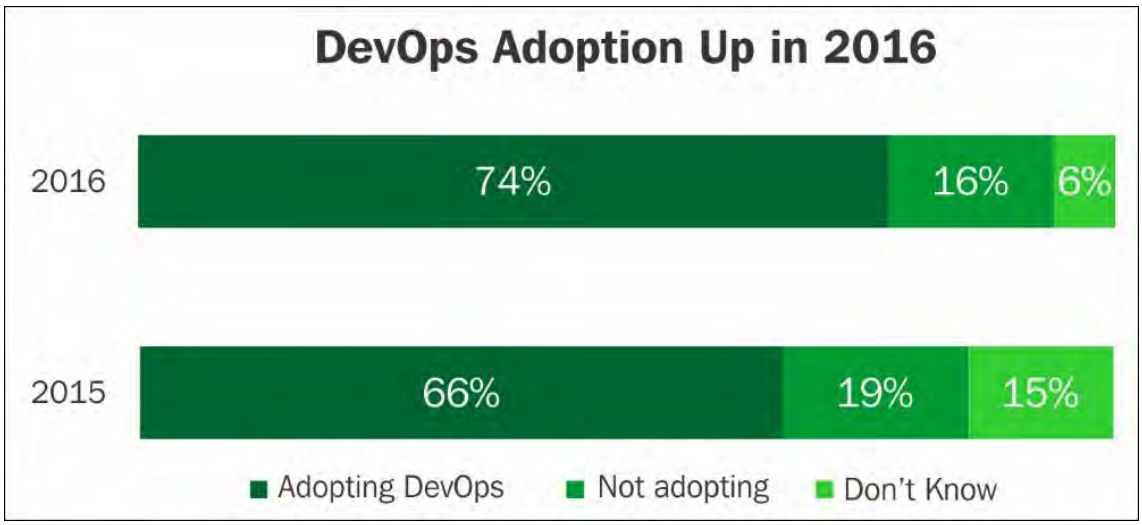


Figura 2: Adopción de DevOps en el año 2016 [38]

En un ambiente de DevOps, la funcionalidad cruzada, las responsabilidades compartidas y la confianza son un valor en alza. DevOps esencialmente extiende el continuo desarrollo de objetivos del movimiento Ágil a una continua integración y lanzamiento. Para hacer más fáciles los lanzamientos, DevOps fomenta la automatización del cambio, configuración y procesos de lanzamiento [7][8].

Los beneficios de este enfoque son muchos, incluyendo [9]:

- Mejora en la frecuencia de instalación, lo cual puede llevar a sacarlo al mercado en un periodo de tiempo más breve.
- Tasa de error más baja.
- Un tiempo de espera más corto.
- Facilita la comunicación entre los desarrolladores, los analistas de calidad y operaciones, mostrándose como un punto de intersección entre los 3 grupos mencionados.

La descripción gráfica de la intersección entre los 3 grupos que interactúan se muestra en la Figura 3.

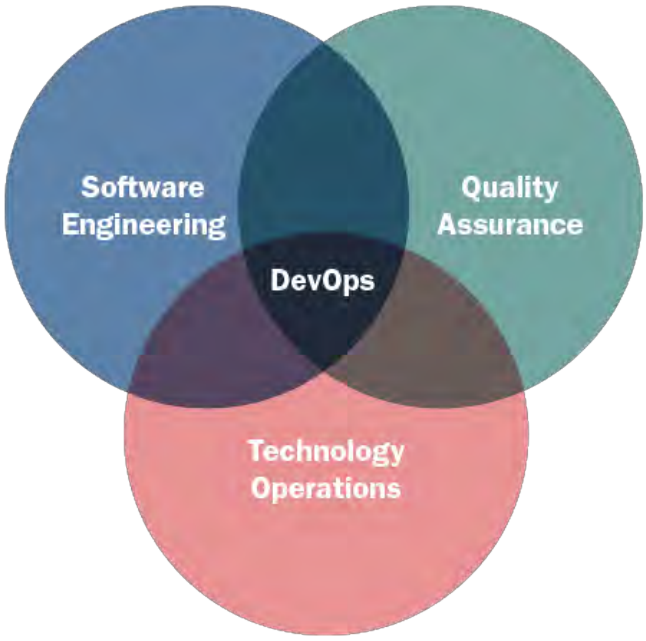


Figura 3: Relación DEVOPS y demás actores [8]

2.5 Integración Continua

El proceso de IC (integración continua) tiene como objetivo principal comprobar que cada actualización del código fuente no genere problemas en una aplicación que se está desarrollando. La integración continua fue utilizada por IBM para el desarrollo del OS/360 en los años 60. [10]

La integración continua no es una herramienta sino más bien una práctica salida del eXtreme Programming (XP) [10].

Los miembros de un equipo de desarrollo integran el programa sobre el que trabajan con frecuencia. La integración continua consiste en activar en cada integración un proceso basado en una plataforma que verifica automáticamente el funcionamiento de la aplicación para que las anomalías sean detectadas lo más pronto posible. [10]

Lo más difícil para el desarrollador es conocer el impacto real de una actualización fundamental sobre todas las funcionalidades de la aplicación. La integración continua permite al desarrollador tener esta visión más global de la aplicación ya que las pruebas de la aplicación se hacen sobre un entorno similar de producción [10].

La práctica de la integración continua lleva varios años ayudando a gestionar mejores proyectos en multitud de organismos debido a que proporciona una gran cantidad de ventajas

como por ejemplo la reducción de tiempo y cantidad de errores en la integración de código, la disponibilidad de la última versión del código a los miembros del equipo, visibilidad de los cambios en código, etc. [10]. Sin embargo, dada la dificultad de uso de las herramientas actuales, aún no es una práctica conocida y utilizada intensivamente en los desarrollos de software [10].

2.6 Entrega Continua

Una vez el software ha alcanzado un grado de madurez adecuado en su fase de desarrollo llega el momento de plantearse que ese software esté disponible para los usuarios finales. Generalmente esto se puede realizar de manera ordenada y planificada, estudiando los riesgos que se van a tomar teniendo claro cómo gestionar las posibles situaciones [11]. Sin embargo, no siempre es posible que sea así, las necesidades de las empresas pueden ser muy diversas y cada vez es más necesario ajustarse el "**time to market",** que tiene mayor presencia en la toma de decisiones y esta política puede introducir variables impredecibles [11]. En muchos contextos se suelen adelantar los plazos de entrega del software, debido a la detección de una oportunidad única, la posibilidad de que un competidor se adelante, u otras consideraciones de negocio, que nada tienen que ver con la planificación y fases del desarrollo del software. En ocasiones es conocido que esta situación se puede dar de antemano o el software se encuentra en un estado donde es probable que sea requerida esta circunstancia [11].

Reconociendo que esta situación puede no ser deseable, pero no por ello deja de ser habitual, se han propuesto metodologías y herramientas que puedan ayudar a instalar el software con el mayor número de garantías posibles, por ejemplo, la metodología "*continous delivery*" (entrega continua). El elemento principal de esta metodología de trabajo es que es necesario que el software esté siempre disponible para poder ser instalado en el entorno accesible a los usuarios en cualquier momento [11].

Se puede construir una versión instalable del software rápidamente, automáticamente y con la disponibilidad de la puesta en producción de los sistemas en cualquier momento, ya sea porque se realiza un cambio para ello, o que exista la necesidad de llevarlo a cabo. Puede realizar instalaciones de cualquier versión del software para cualquier entorno bajo demanda [11].

Se logra la entrega continua, integrando continuamente el software realizado por el equipo de desarrollo, la creación automáticamente de versiones instalables y la ejecución de pruebas automatizadas en esos ejecutables para la detección automática de errores. Además, se deben

poder mover los ejecutables a entornos de producción. Para establecer cuál debe ser el flujo de tareas automatizadas (construcción de los instalables, pruebas automáticas, puntos de chequeos, instalación en diferentes entornos…) se define lo que se denomina un "*Deployment Pipeline*" [11], tal y como se muestra en la figura 2.

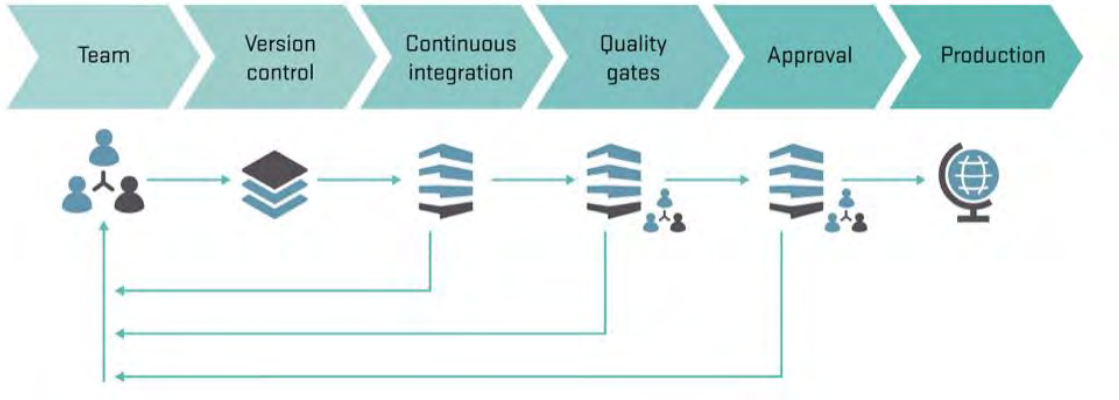


Figura 4: Deployment Pipeline [12]

Para lograr la implementación exitosa de la entrega continua es necesario [12]:

- Una estrecha relación de trabajo colaborativo entre todos los involucrados en la entrega.
- Extensa automatización de todas las partes posibles del proceso de entrega, definido completa y correctamente a través de un *Deployment Pipeline*.

Entrega Continua se confunde a veces con otra metodología llamada "*Continous Deployment*" (despliegue continuo). Despliegue continuo significa que cada cambio pasa por la *Deployment Pipeline* y automáticamente se pone en producción, lo que resulta en muchas instalaciones en producción cada día. Entrega continua sólo significa que podemos ser capaces de hacer despliegues con la frecuencia deseada, pero que se puede optar por no hacerlo, sobre todo atendiendo a la demanda de los usuarios que conocen el negocio con precisión [12].
Se debe tener en cuenta que la entrega continua no es únicamente la instalación en producción, sino la puesta en producción pasando por procesos de validación y calidad automatizados. El objetivo es por tanto que la versión que se está instalando en producción se haga de manera rápida (sin necesidad de una planificación especial), se haga de manera eficaz (sin afectar a equipos con otras tareas planificadas) y con todas las garantías posibles (todas las comprobaciones que se puedan llevar a cabo de manera automática y asegure que

no se están introduciendo nuevos errores) [16]. En muchas empresas con gran parte de su negocio informatizado la instalación de software de manera urgente (algo más frecuente de lo que cabe esperar) suele ser un gran esfuerzo para muchos equipos implicados, afectar a planificaciones y no siempre se puede realizar con todas las garantías adecuadas. [11]
Una manera de comprobar que la entrega continua está correctamente implantada en una empresa es cuando un usuario de negocio solicita el despliegue de la versión actual y nadie entra en pánico. [12]
Una vez comentada la metodología es importante tener en cuenta que existen herramientas que van a ayudar a automatizar las tareas y construir (y mantener) el *Deployment Pipeline*. Algunas de estas herramientas son:

- **TeamCity**: es una de las herramientas de integración continua más populares. Gracias a su versatilidad y la gran cantidad de *plugins* que tiene es posible adaptarla para ser empleada como herramienta esencial para utilizar entrega continua [14]. Sin embargo, está centrada en integración continua y tiene importantes limitaciones para ser empleada como entrega continua, tales como no tener claro el concepto de *Deployment Pipeline*.
- **Octopus**: Es una herramienta completamente diseñada para realizar despliegues. Permite definir de manera clara y concisa el *Deployment Pipeline*. Está diseñada para integrarse con otras herramientas de distintas fases que es necesario para llevar a cabo la entrega continua de manera exitosa. [15]

A parte de las herramientas citadas es necesario que exista un ecosistema de herramientas que permitan facilitar la automatización de tareas que se deben realizar [11].
La entrega continua es una herramienta imprescindible para muchas empresas que tienen gran parte de su negocio informatizado. La actualización de software es un elemento crítico para estas empresas, por tanto, aplicar esta metodología de manera exitosa supone una ventaja competitiva respecto a empresas cuyos protocolos no están lo suficientemente automatizados o no cumplen todos los pasos necesarios, ya que permite formalizar procesos, establecer puntos de chequeo precisos y estandarizar procesos en situaciones que pueden ser críticas [12].
Algunas grandes empresas como Amazon, Google o Twitter (entre otros) llevan años aplicando esta metodología con resultados exitosos [11]. La divulgación de esta metodología, la proliferación de herramientas para gestionarla mejor está haciendo que se vaya extendiendo

a otras empresas de menor tamaño que están consiguiendo los beneficios de esta metodología [12].

2.7 Pruebas

2.7.1 Metodologías para Pruebas

Existen diferentes aproximaciones o metodologías que orientan el diseño de las pruebas. Cada metodología puede usarse con los diferentes tipos de pruebas e incluso combinarse para adaptarse a las necesidades de los proyectos [13].

- **Caja Negra:** Se evalúan los componentes como una caja negra. Solo las entradas y salidas [13].
- **Caja Blanca:** Se evalúan los estados internos y llamados internos a otros componentes [13].
- **Caja Gris:** Es una mezcla entre el método de caja blanca y caja negra. Básicamente se evalúa la especificación de alto nivel del sistema y los componentes y llamados internos [13].
- **Adhoc:** Es una metodología de pruebas donde se validan los componentes sin documentación o especificación. El Ingeniero de calidad trata de encontrar bugs sin ningún plan basándose únicamente en su intuición [13].

Para el presente proyecto se utilizó las pruebas de caja negra, debido a que su ejecución considera las pruebas de los requisitos funcionales.

2.7.2 Tipos de Pruebas

A continuación, se presenta una lista de las pruebas realizadas dentro de la empresa. Cada prueba forma parte del aseguramiento de calidad del producto de software a ser entregado.

- **Pruebas de Unidad**

  Este tipo de pruebas validan que una unidad de código (una clase, función o método) realice el trabajo para el que fue diseñado. Son fragmentos de código que validan la ejecución de las unidades de código del producto que valida código. Opcionalmente pueden validar los llamados a otros sistemas o estados intermedios [13].

- **Pruebas de Integración**

Las pruebas de integración validan el funcionamiento o integración entre dos o más componentes. Por ejemplo, el acceso a un servicio web o el almacenamiento en la base de datos. El objetivo de las pruebas de integración es validar el comportamiento de múltiples componentes. Dependiendo de la implementación pueden validar estados intermedios [13].

- **Pruebas de Regresión**

Las pruebas de regresión son las pruebas que se ejecutan después de un cambio en el sistema (nueva característica o solución a un bug). Su objetivo principal es validar que las características previas al cambio no se vean afectadas. Son las primeras candidatas a automatización pues deben repetirse periódicamente con cada uno de los cambios o actualizaciones [13].

- **Pruebas de Humo**

Este tipo de pruebas son pruebas rápidas que buscan encontrar problemas graves (fuego oculto) en el software recientemente modificado. El nombre viene de los sistemas electrónicos donde la prueba inicial es que no hay humo al conectar la fuente de energía. Las pruebas de humo buscan problemas críticos en la aplicación que hacen que no valga la pena validar otro tipo de pruebas. Preguntas como: ¿El sistema inicia correctamente? o ¿El menú principal se presenta correctamente? caracterizan el objetivo de las pruebas de humo [13].

- **Pruebas de Aceptación**

El objetivo de las pruebas de aceptación no es validar el comportamiento lógico específico de un componente, sino un escenario, caso de uso o caso de negocio de la aplicación. Permite validar que la aplicación realiza las tareas para las que fue diseñada [13].

- **Pruebas de Sistema**

Las pruebas de sistema validan el funcionamiento del software en condiciones específicas (sistemas operativos, navegadores, servidores, bases de datos o tipo dispositivo). Las pruebas de sistema también pueden validar la escalabilidad, el desempeño y la confiabilidad del sistema en escenarios específicamente diseñados [13].

2.8 Automatización de Pruebas

La automatización de pruebas de software se ha movido más allá de un lujo para convertirse en una necesidad [17]. Las aplicaciones y los sistemas han crecido cada vez más y más complejos, y las pruebas manuales simplemente no pueden mantenerse al día. A medida que la tecnología cambia y más organizaciones se mueven hacia un desarrollo ágil, las pruebas deben adaptarse y rápidamente. La automatización de pruebas es esencial [17].

Este es el motivo por el que la automatización de pruebas es una recomendación, aunque no una obligación, útil para crear un entorno de desarrollo satisfactorio [17]. Los conjuntos de casos de prueba garantizan que la aplicación hace lo que se supone que debe hacer. Incluso cuando el código interno de la aplicación cambia constantemente, las pruebas automatizadas permiten garantizar que los cambios no introducen incompatibilidades en el funcionamiento de la aplicación. Además, este tipo de pruebas obligan a los programadores a crear pruebas en un formato estandarizado y muy sólido que pueda ser procesado por un framework de pruebas [17].

2.9 Reducción de Costos

Como se puede observar en la Figura 3, cuanto más se tarda en detectar un error, más costosa será su reparación [18]. Por medio de la Integración Continua los errores de integración de código pueden ser detectados en una etapa anterior a la etapa de pruebas [4].

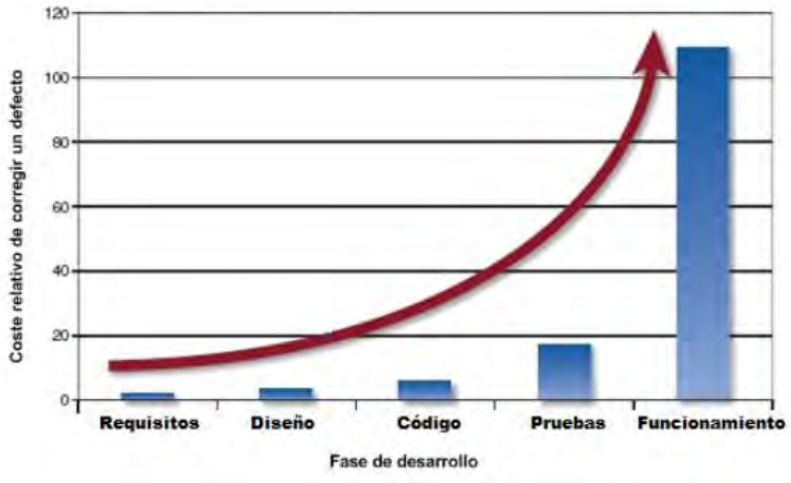


Figura 5: Coste de corregir un defecto según la fase de desarrollo [18]

Existen otros datos económicos que justifican el uso de la IC. Barry Boehm, uno de los mayores expertos en calidad del software, ha publicado varios estudios que demuestran cómo el coste de eliminar un defecto en un software crece exponencialmente en cada fase en la que aún no se haya descubierto dicho fallo [19]. Otros trabajos han confirmado las teorías de Boehm: En [20] se explica que reparar un defecto durante la etapa de desarrollo de un módulo podría tener un coste aproximado de un dólar. Sin embargo, la reparación costaría más de cien dólares cuando ya se ha integrado todo el código y miles de dólares si el defecto se detecta cuando el software ya se ha distribuido. Un estudio del NIST ("*National Institute of Standards and Technology*" o Instituto Nacional de Estándares y Tecnologías) ha revelado que los defectos de software hacen que en los Estados Unidos se gasten 60 billones de dólares cada año. También el NIST ha reconocido que casi el 80 % del coste total de los proyectos se destina a corregir sus defectos [21].

CAPÍTULO III: ESTADO DEL ARTE

Este capítulo tiene como objetivo ubicar y clasificar estudios previos sobre los lenguajes de especificación usados en la automatización de pruebas y las herramientas que soportan la implementación de un desarrollo guiado por comportamiento (Behavior Driven Development - **BDD**), para lo cual se llevó a cabo una revisión sistemática y el estudio de las herramientas existentes en el mercado.

3.1 Introducción

Una revisión sistemática de la literatura técnica existente en cualquier área del conocimiento científico es muy importante, ya que permite identificar, evaluar, interpretar y sintetizar las investigaciones existentes relevantes en un tema de interés en particular. No solo esto, sino que permite resumir las evidencias existentes reconociendo sus beneficios y limitaciones, así como identificar algunos vacíos que puede resultar importante investigar con mayor detenimiento [24].

Para llevar a cabo la revisión sistemática se tomó como referencia la plantilla del protocolo presentado en [24], el cual consta de cinco partes generales: formulación de la pregunta, selección de las fuentes, selección de los estudios, extracción de información y resumen de los resultados.

3.2 Formulación de la pregunta

Las preguntas de investigación son:

- ¿Qué lenguaje de especificación de pruebas de aceptación es recomendado en el desarrollo guiado por comportamiento **(BDD)**?
- ¿Qué herramienta es recomendada para la automatización de pruebas de aceptación en un desarrollo guiado por comportamiento (**BDD**)?

Las palabras claves empleadas en la estrategia de búsqueda para resolver la pregunta de investigación fueron: *testing, test, "behavior-driven development"*.

Los resultados esperados al finalizar la revisión sistemática son, entre otros, los estándares para llevar a cabo el desarrollo guiado por comportamiento y las herramientas y/o

frameworks para realizar su implementación. La población para observar fueron las publicaciones relacionadas con automatización de pruebas y BDD en las fuentes seleccionadas para la revisión.

3.3 Selección de Fuentes

A partir de los términos seleccionados: *"testing", "test" y "behavior-driven development"* combinados con el conector lógico AND se obtuvieron las cadenas generales de búsqueda como se establecen en la Tabla 1

N	Cadena general de búsqueda
1	"behavior-driven development" AND (test OR testing)

Tabla 1: Cadenas Generales de Búsqueda

Las fuentes con las que se ejecutó la revisión sistemática fueron principalmente ACM Digital Library, IEEE Xplore, también se consultaron Science Direct, Scopus y Google mismo. En el caso de la última, las referencias encontradas son muchas pero las fuentes referenciadas no garantizan trabajos confiables por lo que se tomaron de forma referencial. Lo que se debe señalar es que Google es de gran ayuda para ubicar los documentos referenciados por las otras fuentes a los que no se tiene acceso para obtener los artículos.

3.4 Selección de los estudios

El criterio de la ejecución de la revisión sistemática fue basado en el modelo iterativo e incremental. Iterativo porque la ejecución (búsqueda, extracción de información y visualización de resultados) de la revisión sistemática se hace primero completamente en una fuente de búsqueda, y así sucesivamente sobre las demás. Incremental porque el resultado de la revisión sistemática va creciendo y evolucionando en cada iteración hasta convertirse en definitivo [24]. De esta forma se obtuvieron todas las ventajas que ofrecen este tipo de modelos de desarrollo.

Como criterios de inclusión para la selección de estudios primarios se ha considerado:

- Artículos publicados en inglés;
- Fecha de publicación a partir del año 2007

- Se realizó la búsqueda en artículos publicados en revistas y publicaciones realizadas en Congresos.
- Artículos que versen sobre herramientas usadas para la implementación de las pruebas en el desarrollo guiado por comportamiento.

La inclusión de los estudios encontrados también se basó en el análisis del título, resumen y palabras claves de los artículos encontrados en la búsqueda.
Para determinar qué artículos relevantes eran suficientemente importantes en el contexto de la revisión sistemática para ser considerados como estudios primarios se definió como criterio de exclusión de los estudios a todos los estudios que estaban orientados al desarrollo y no a las pruebas. También se excluyeron los trabajos que no cuenten con una herramienta para la implementación de las pruebas.

En el proceso de la selección de los estudios primarios se encontraron 54 artículos. No se encontraron artículos duplicados, lo que resulta en una población de 54 artículos. Para realizar el primer filtro se aplicaron los criterios de inclusión considerando el título y resumen de los trabajos. Como resultado se obtuvieron 22 trabajos candidatos.
Finalmente, en una segunda etapa se seleccionaron los 6 artículos primarios a partir de la lectura completa de los trabajos anteriores, verificando que efectivamente satisfacían los criterios especificados.

Base de Datos	**Relevantes**	**Candidatos**	**Primarios**
IEEE Xplore	22	10	3
Science Direct	23	8	1
ACM Digital	6	4	1
Scopus	3	0	0
Total	54	22	5

Tabla 2: Selección de artículos primarios

La tabla 2 muestra la relación de artículos. De los trabajos primarios seleccionados, destaca que más de la mitad (60%) han sido publicados en IEEE Xplore.

3.5 Extracción de Información

Una vez escogido los estudios primarios que puedan resolver las preguntas definidas anteriormente se realizó la extracción de la información relevante para la revisión sistemática.

3.5.1 Lenguaje de Especificación para Pruebas de aceptación

Tiayao Li et al. [27] aplica una automatización de pruebas usando un estilo de desarrollo guiado por comportamiento. Presenta un problema en el tamaño de las configuraciones a ser probadas, la cual es resuelta parametrizando dichas configuraciones. Se llega a concluir que Gherkin es un lenguaje de especificación de pruebas que puede llegar a moldearse, de manera que soporte parametrizaciones de dicha magnitud.

En tanto Mazedur Rahman et al. [32] propone una arquitectura reusable para la automatización de pruebas de aceptación en micro servicio utilizando el desarrollo guiado por comportamiento. Se llega a la conclusión que la llave del éxito en dicho entorno, son las pruebas de aceptación ejecutables descritas usando una sintaxis simple, y que pueda ser entendido de manera rápida y sencilla. En este artículo se presenta a Gherkin como dicha sintaxis.

Así mismo Michael Kart [25] en el estudio que hace sobre el desarrollo guiado por comportamiento mismo, propone técnicas efectivas para su implementación usando un lenguaje Gherkin.

Por su parte Ioa Lazar et al. [33] propone un lenguaje de especificación para pruebas diferente. Con el motivo de construir escenarios ejecutables que sigan el criterio de aceptación en las pruebas se utiliza "*Foundational UML*" (fUML). Los desarrolladores y los expertos en el dominio de los requerimientos deben tener un lenguaje común de manera q represente el comportamiento deseado del sistema.

Con respecto al lenguaje de especificación para la realización de las pruebas de aceptación se observa que el más utilizado es el Lenguaje Gherkin, el cual ofrece muchas más ventajas sobre (fUML), al ser un lenguaje de fácil configuración, sencillo y de ágil desarrollo.

3.5.2 Herramienta (framework) para automatización de Pruebas de aceptación

Nan Li et al. [28] tiene por objetivo la implementación de pruebas de aceptación automatizadas, usando una metodología de desarrollo guiado por comportamiento, la herramienta (framework) con la cual se implementó la automatización es "Cucumber".

Cucumber, una de las herramientas más utilizadas en el mercado, permite el uso de un lenguaje común, que puede ser entendido por los analistas del producto, desarrolladores y los analistas de calidad [28].

Con respecto a la herramienta o framework para la automatización de las pruebas de aceptación la información encontrada en los trabajos revisados es muy limitada, sin embargo, se observa las ventajas y facilidades que ofrece Cucumber como herramienta.

3.6 Herramientas Existentes

3.6.1 Herramientas para la Implementación de Automatización

Existen diferentes herramientas que soportan instrucciones en lenguaje natural (Gherkin). Cada una de estas herramientas adopta diferentes implementaciones en diferentes lenguajes de programación:

- Cucumber para Ruby
- JBehave para Java
- NBehave y SpecFlow para C#
- Freshen para Python
- Behat para PHP

Todos ellos tienen algunas características comunes admitidas, pero existen algunas restricciones y habilidades específicas para el motor en sí. Por lo tanto, se recopiló características útiles para cada motor enumerado anteriormente y se presentó en forma comparable. Las principales características evaluadas son:

1. Disponibilidad de documentación
2. Flexibilidad en pasar parámetros

3. Autocompletar
4. Pasos, escenario y definición de características
5. Pasos complejos
6. Ganchos y condiciones previas
7. Encuadernación al código
8. Flexibilidad de formateo
9. Informes incorporados
10. Soporte de fuentes de datos de entrada

Cada característica tiene algunas características secundarias que reflejan alguna parte específica de la funcionalidad. Y esas pequeñas cosas hacen que cada motor sea diferente y único en algunos casos. Para poder compararlas se introdujo la escala de calidad de soporte.

Grado	Criterio
0	No Soporta.
1	Funcionalidad existe, pero con restricciones considerables.
2	Funcionalidad principal existe.
3	Soporte con todas las funciones.

Tabla 3: Criterios de calificación [Elaboración Propia]

Cada característica puede ser valorada con la calificación de 0 a 3 según los criterios propuestos. Es así como se evaluó cada herramienta con las 10 características mencionadas.

	Característica										
Herramienta	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Cucumber	3	3	1	3	3	3	2	3	3	0	**24**
Freshen	2	2	1	0	3	3	3	3	2	2	**21**
JBehave	3	2	1	3	3	1	3	1	2	3	**22**
NBehave	1	2	1	0	2	0	3	3	2	0	**14**
SpecFlow	3	2	3	3	2	2	2	2	2	0	**21**
Behat	3	3	1	3	3	3	2	3	1	0	**22**

Tabla 4: Evaluación de Herramientas [Elaboración Propia]

Se puede observar la superioridad existente de Cucumber sobre otras herramientas como: Fresehn, JBehave, NBehave, SpecFlow, Behat.

3.6.2 Herramientas de Automatización Empresariales

En la actualidad existen diferentes empresas que ofrecen el servicio de automatización de pruebas, así como herramientas para cumplir dicho propósito.

Los proveedores de herramientas de automatización con más participación en conferencias sobre tecnología y Software Testing (StarWEST, InfoQ, AgileWEST, etc.):

- Saucelabs
- TestDroid
- Xamarin Test Cloud
- BrowserStack
- CrossBrowserTesting
- Parasoft

Así como los manejadores de casos de prueba (*test case management*) más populares que poseen integración con JIRA [14]:

- Zephyr
- Hiptest (*Supports BDD*)
- TestRail
- XRay (*Supports BDD*)

Los administradores de casos de prueba Hiptest y XRay proveen soporte a BDD, y los casos de prueba pueden ser escritos con lenguaje Gherkin.

3.7 Resultados y Discusión

La revisión realizada ayuda a determinar el lenguaje y la herramienta a utilizar en la automatización de las pruebas de aceptación.

Se determinó que el lenguaje recomendado en la creación de las pruebas de aceptación es el lenguaje GHERKIN, el cual será tomado en cuenta como información de entrada (INPUT) en el sistema, el cual estará sujeto a automatización.

Si bien es cierto no hay una variedad de estudios sobre herramientas que implementan la automatización de los casos de prueba, se concluye que CUCUMBER posee ventajas sobre cualquier otro similar, y es la seleccionada como base para la implementación de la automatización dentro del sistema propuesto.

Las herramientas de automatización existentes en el mercado ofrecen su propia plataforma que permite automatizar pruebas y obtener reportes de los resultados. El problema con el uso de dichas herramientas es que hacen adecuar el proceso establecido por la empresa al uso de la herramienta. Dichas herramientas no proveen soporte a un enfoque orientado a BDD.

Solo 2 administradores de casos de prueba con soporte en jira proveen un soporte para el lenguaje Gherkin, sin embargo, la implementación de la automatización misma tiene que ser realizada de manera aislada fuera de la herramienta.

La herramienta propuesta se adecúa al proceso de desarrollo y pruebas establecido, teniendo como base el uso de JIRA para el manejo de los planes de prueba.

CAPÍTULO IV: EJECUCION DE LA INVESTIGACION

En el presente capitulo se presenta el desarrollo de la tesis, centrado en las actividades de la fase de aceptación de pruebas en el proceso de desarrollo.

4.1 Introducción

La ejecución de la investigación es acerca el desarrollo de un sistema web para automatizar la fase de pruebas de aceptación en la empresa de desarrollo de software.

4.2 Análisis del Proceso

4.2.1 Proceso Desarrollo

El proceso actual de desarrollo basado en metodologías agiles (Scrum & Kanban) consta de las siguientes etapas, tal como se muestra en la Figura 5:

- Recibimiento de Requerimientos basado en Historias de Usuario.
- Estimación de Requerimientos basado en horas hombre.
- Desarrollo de Requerimientos
- Integración de los cambios con Código Principal
- Corrida de Pruebas Unitarias
- Despliegue del sistema en ambientes de QA
- Realización de Pruebas de Aceptación en ambiente de QA
- Despliegue del sistema en ambiente de UAT
- Realización de Pruebas de Aceptación en ambiente de UAT
- Certificación del cambio por parte del equipo de Calidad

Para llevar un mejor control de las etapas del proceso se usan diferentes herramientas proporcionadas y licenciadas en la empresa:

- **BitBucket**, herramienta de Atlassian para controlar las versiones del código fuente.
- **TeamCity**, herramienta de JetBrains utilizada para la integración continua de cambios y para correr las pruebas unitarias de manera automatizada.
- **Octopus**, herramienta utilizada para poder desplegar el sistema en cualquier ambiente de pruebas y puesta en Producción.

- **Jira**, herramienta de Atlassian utilizada para llevar un control de los requerimientos, los planes de desarrollo por requerimiento, así como los planes de testeo por requerimiento.

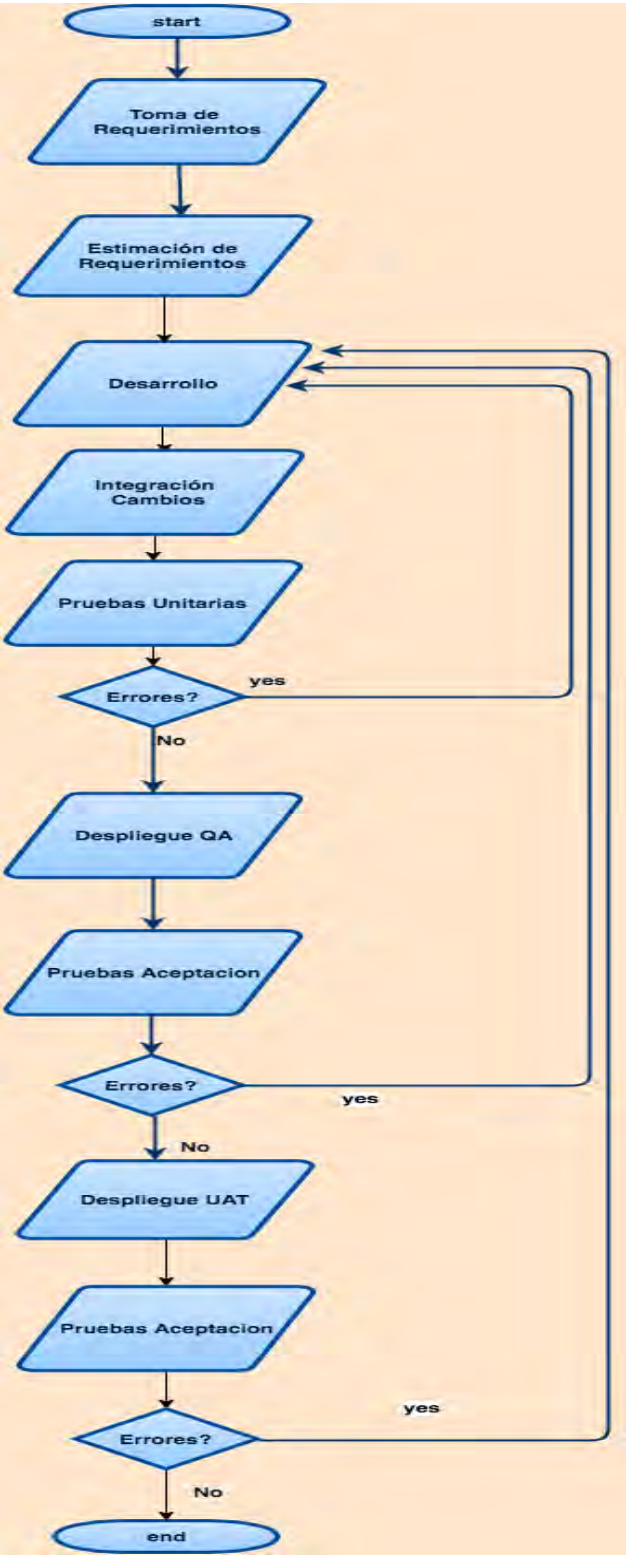


Figura 6: Flujo Proceso de Desarrollo [Elaboración Propia]

Artefactos usados en el proceso de desarrollo, presentados en la Figura 5:

- ***Client Initiative***, tipo de *issue* utilizado en jira para llevar un control de los proyectos solicitados del cliente.
- ***Requirement***, tipo de issue utilizado para describir en detalle los requerimientos dentro de un *Client Initiative*.
- ***Dev Plan***, tipo de *issue* utilizado para llevar control del desarrollo de código hecho (cada *commit* en el código es relacionado a este tipo de *issue*).
- ***Test Plan***, tipo de *issue* utilizado para llevar un control de las pruebas de aceptación realizadas a cada *Client Initiative*.
- ***Cignium Defect***, tipo de *issue* utilizado para registrar defectos encontrados en el ambiente de QA, realizado por los analistas de calidad de Cignium.
- ***UAT Defect***, tipo de *issue* utilizado para registrar defectos encontrados en el ambiente de UAT, realizado por los analistas de calidad de DirecTV.

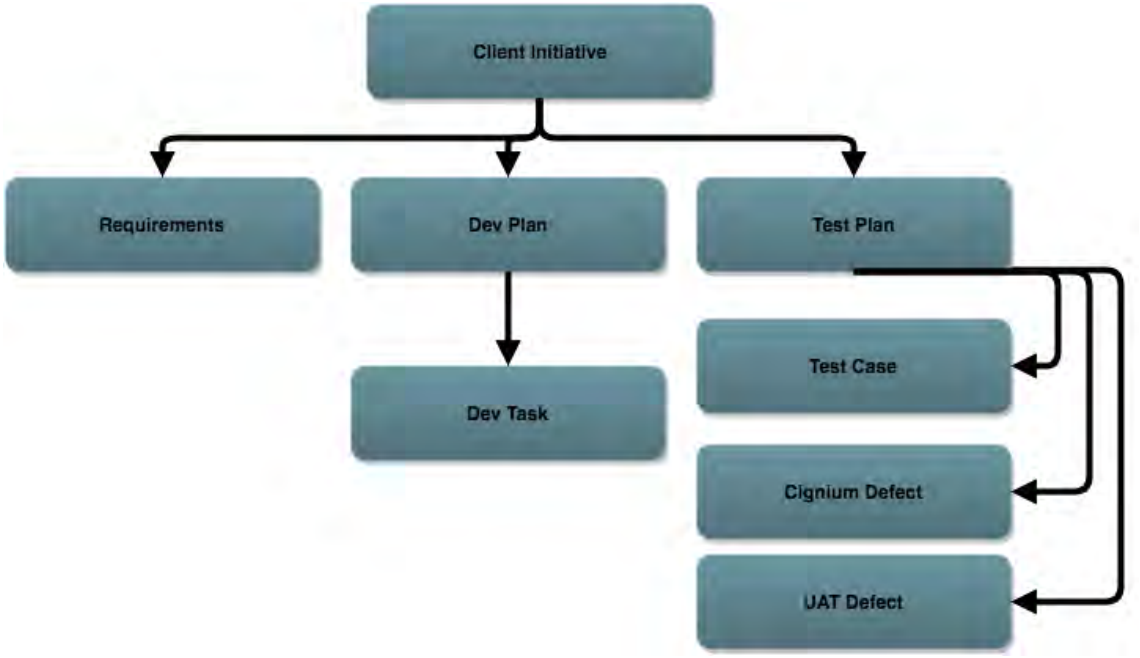


Figura 7: Artefactos en el proceso de Desarrollo [Elaboración Propia]

4.2.2 Realización de Pruebas de Aceptación

Las pruebas de aceptación incluyen pruebas realizadas por los analistas de calidad en la empresa de desarrollo de software y por los analistas de calidad de DIRECTV en los ambientes de No-Producción de QA (donde el sistema es desplegado 2 veces al día) y UAT

(donde el sistema es desplegado 2 veces a la semana) respectivamente. En el ambiente de UAT, se despliega la versión del sistema que ha sido probada exitosamente en el ambiente de QA.

Las pruebas de aceptación son realizadas en el ambiente de QA y UAT hasta que son satisfactorias. Se realizó un marco de trabajo para la realización de las pruebas de aceptación, el cual se encuentra detallado en el Anexo 9.
El problema encontrado en esta etapa radica en que el sistema está sujeto a cambios después que las pruebas ejecutadas sean satisfactorias, estando sujeto a que un cambio en un requerimiento diferente cambie el comportamiento deseado en el sistema.
En caso de encontrarse error, la prueba será repetida n veces, hasta que el test del requerimiento sea exitoso.

4.3 Desarrollo del Sistema Web para Automatización

El proyecto de software definido será desarrollado e implementado usando la metodología XP (extreme Programming), para lo cual se define las siguientes etapas:

- Exploración
- Planificación de las entregas
- Iteraciones
- Producción

4.3.1 Exploración

En esta etapa se analizó la problemática, se conversó con los analistas de calidad para el planteo de las funcionalidades requeridas por el sistema. Por lo que se obtuvo:

- Historias de Usuarios
- Análisis de Herramientas y estándares por usar.
- Prototipos de Pantalla
- Se generó un diagrama de arquitectura del Sistema

a. Historias de Usuarios

Se usó la técnica más utilizada en el desarrollo de proyectos de software que usan metodologías agiles [23], obteniendo una lista de historia de usuarios como se muestra en la Tabla 1.

N	Nombre de la Historia	Usuario Rol
1	Visualización de Test Plan y Test Cases	Analista Calidad
2	Creación y Edición código Gherkin	Analista Calidad
3	Ejecución de Test Cases	Analista Calidad
4	Visualización de Logs de la Ejecución	Analista Calidad
5	Generación de Reportes de Ejecución	Analista Calidad
6	Autenticación *"Active Directory"*	Analista Calidad
7	Ejecución en Chrome y Firefox	Analista Calidad

Tabla 5: Historias de Usuario

b. Herramientas

Se estableció el marco de desarrollo donde se definió las herramientas a usar en el proyecto de desarrollo, así como la tecnología y la arquitectura a utilizar.

Tecnologías Utilizadas:

- WebApi en .Net
- MVC y Bootstrap
- C# en backend
- SignalR
- Ruby para programación de pruebas
- Cucumber para escritura de pruebas
- MongoDb como base de datos
- Dockers y Containers para la ejecución de pruebas
- Jira APIs para obtener los Test Plan y Test Cases

Las herramientas utilizadas:

- Visual Studio 2017
- RubyMine
- Azure

4.3.2 Planificación de las Entregas

En esta fase se realizó un trabajo de identificación y estimación tanto de las historias y sus prioridades como del tiempo que tomará su desarrollo. Se definió las iteraciones y qué

historias se desarrollaron en cada iteración, como los entregables y qué iteraciones lo conformarán, como se muestra en la Tabla 2.

a. Priorización

El proyecto por entregar consta de un solo entregable con 3 iteraciones.

N	Nombre de Historia	Prioridad	Riesgo	Esfuerzo	Iteración
1	Visualización de Test Plan y Test Cases	Alta	Medio	3	1
2	Creación y Edición código Gherkin	Alta	Bajo	3	2
3	Ejecución de un Test Cases	Alta	Bajo	3	2
4	Visualización de Logs de la Ejecución	Baja	Bajo	2	3
5	Generación de Reportes de Ejecución	Media	Bajo	2	2
6	Autenticación *"Active Directory"*	Baja	Bajo	1	3
7	Ejecución en Chrome y Firefox	Baja	Bajo	1	2

Tabla 6: Historias de Usuario con Prioridad

4.3.2 Iteraciones

Es en esta parte donde se tiene el proyecto dividido en unidades de trabajo, y se describe las pruebas que se deben realizar para mantener un código libre de errores.

a. Primera Iteración

Según lo planeado se debe desarrollar las historias:

- H1: Visualización de Test Plan y Test Cases

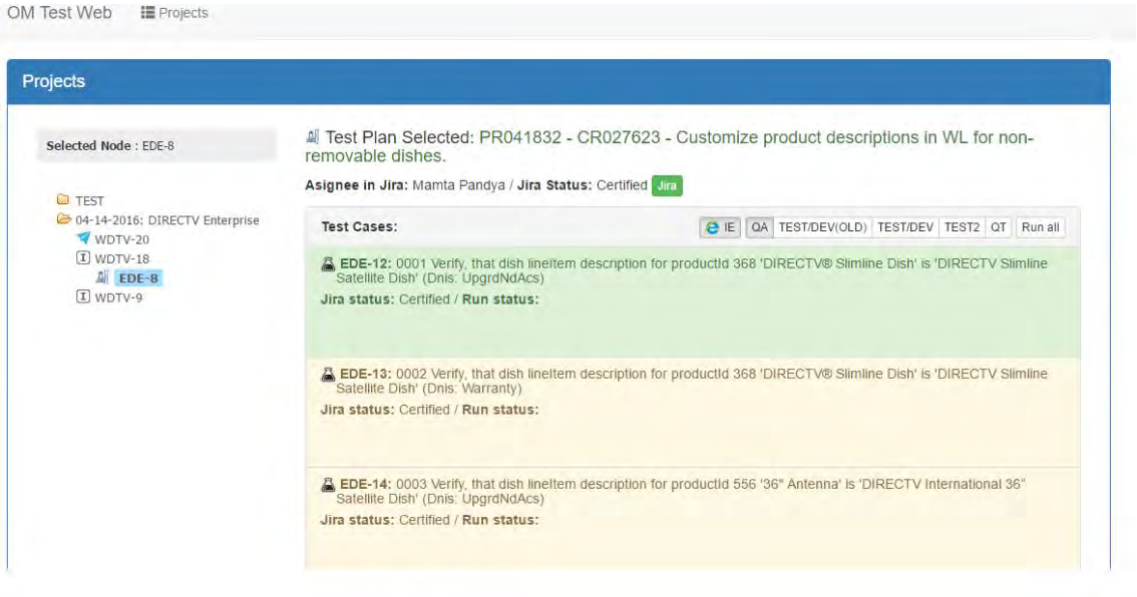


Figura 8: UI en la primera iteración

b. Segunda Iteración

Según lo planeado se debe desarrollar las historias:

- H2: Creación y Edición código Gherkin
- H3: Ejecución de un Test Cases
- HS5: Generación de Reportes de Ejecución
- HS7: Ejecución en Chrome y Firefox

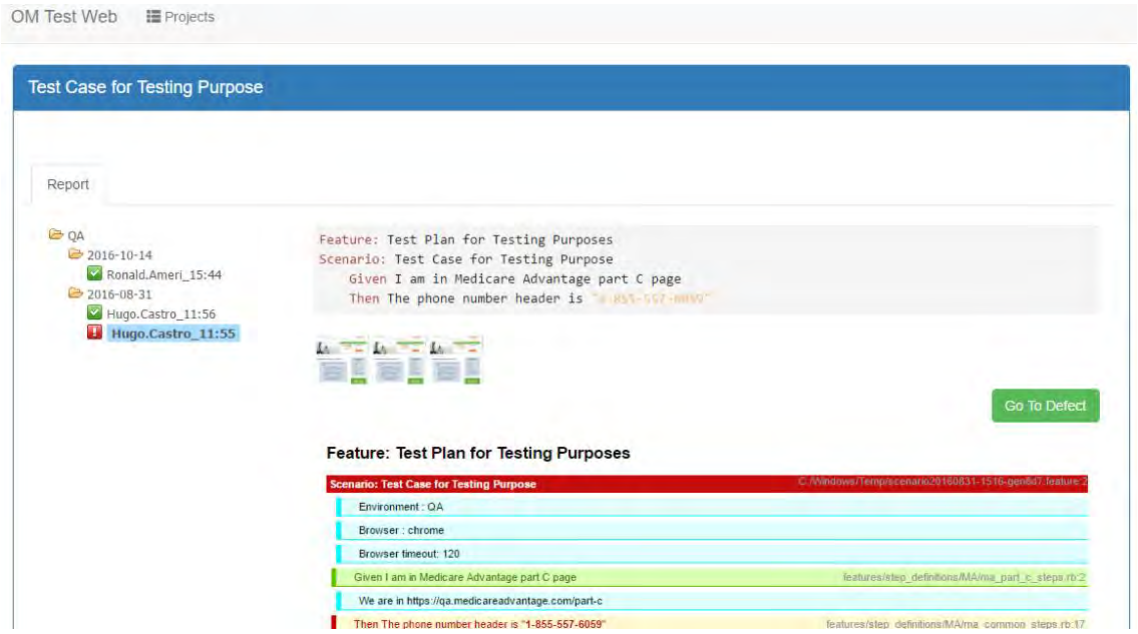


Figura 9: UI en la segunda iteración: Reportes

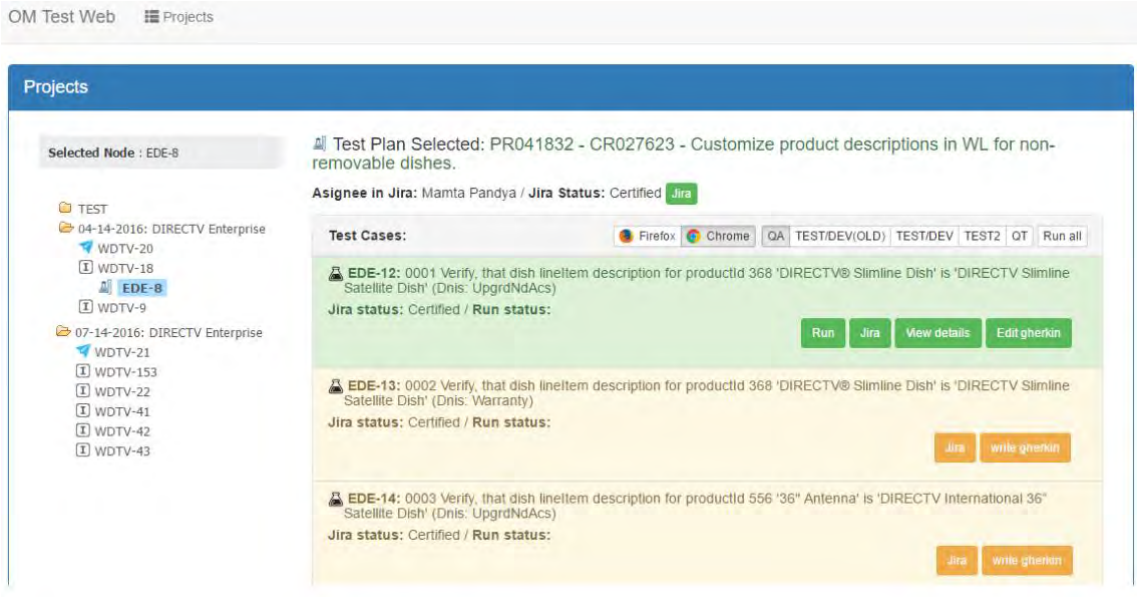


Figura 10: UI en la segunda iteración: Ejecución

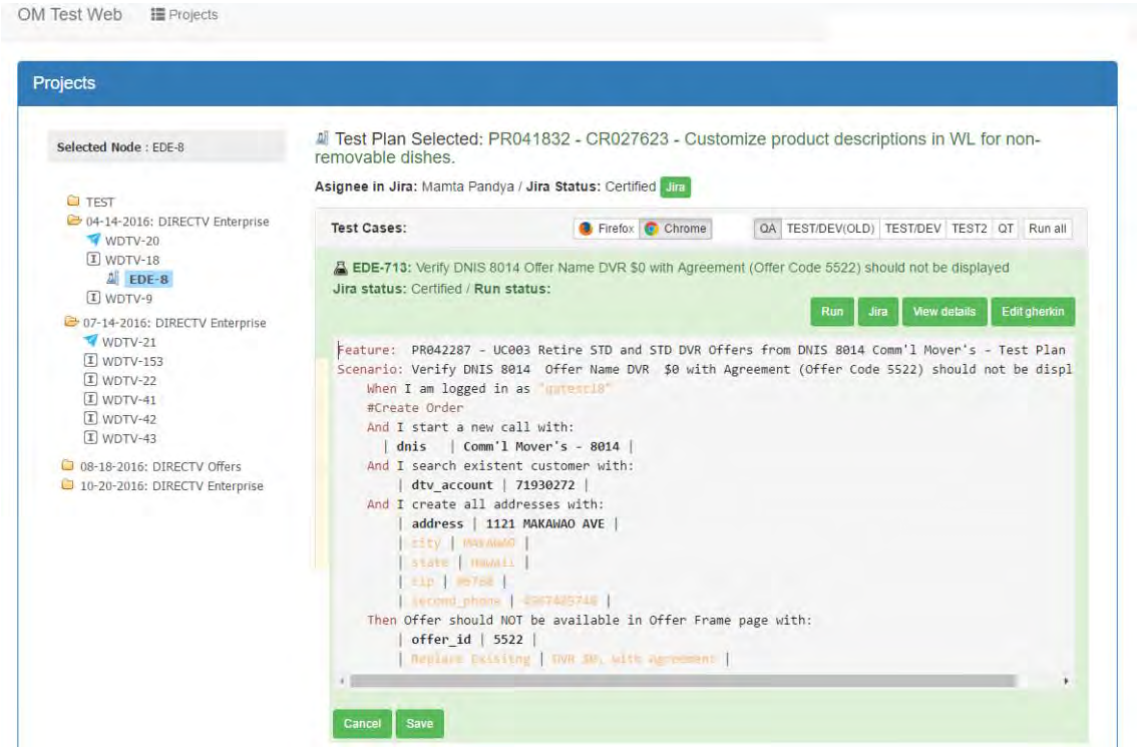


Figura 11: UI en la segunda iteración: Creación y Edición Gherkin

c. Tercera Iteración

Según lo planeado se debe desarrollar las historias:

- H4: Visualización de Logs de la Ejecución
- H6: Autenticación Active Directory

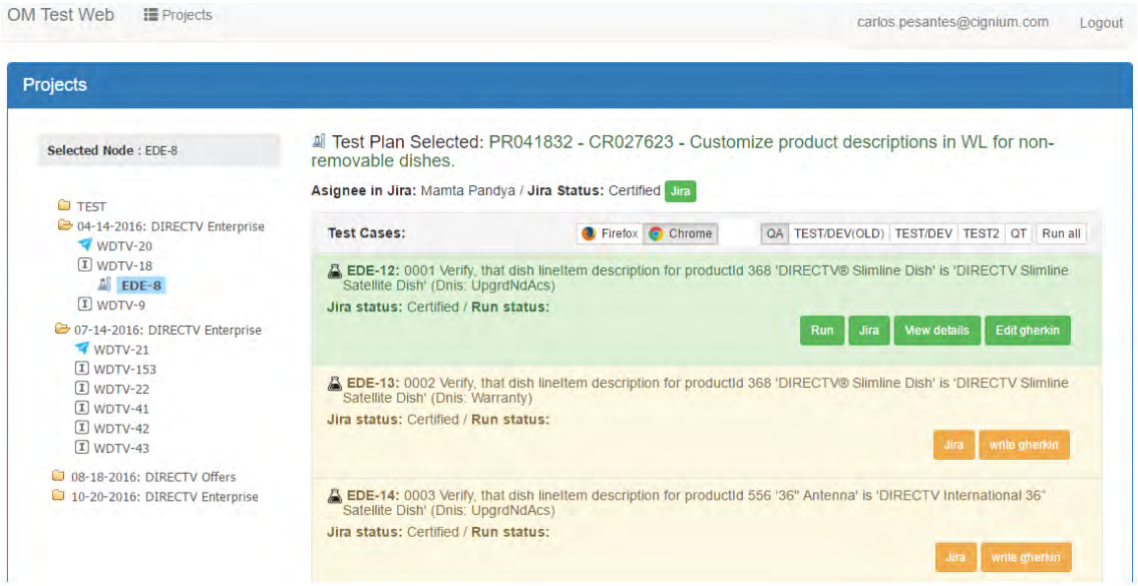


Figura 12: UI en la tercera iteración: Active Directory

Figura 13: UI en la tercera iteración: Logs

4.3.2 Producción

En este paso se pasa a promover el sistema en los servidores de Azure proporcionados por la empresa.

Actualmente se cuenta con 2 virtual server:

- Windows Virtual Box: Hosteo del Website y el WebApi
- Linux Virtual Box: Hosteo de los dockers y containers donde se ejecutan las pruebas.

4.4 Métodos

La toma de muestra se realizó a través del uso de 3 sistemas en conjunto:

a) Jira

Para obtener la población se realizaron búsquedas avanzadas que permiten utilizar consultas estructuradas (Anexo 8). Se utilizó el lenguaje de consultas de jira (JQL o JIRA query Language).

b) Unanet

Para obtener el tiempo empleado para la realización de las pruebas de aceptación de cada proyecto se utilizó los reportes generados por unanet (Sistema de control de horas trabajadas).

4.4.1 Pruebas de Normalidad

Se realizaron las pruebas de normalidad:

- Kolmogorov-Smirnov, se utilizó cuando el tamaño de la muestra es mayor a 50,
- Shapiro-Wilk, se utilizó cuando el tamaño de la muestra es menor a 50.

Hipótesis Estadística

- Hipótesis Ho

  Ho, La variable tiene distribución normal.

- Hipótesis H1

  H1, La variable es distinta a la distribución normal.

Se aplicó las pruebas de normalidad en las muestras de las variables de los 6 indicadores, posteriormente de comprobar que las muestras siguen una distribución normal, se utilizó las pruebas paramétricas t Student (diferencia de medias) y Z (diferencia de medias).

4.4.2 Diseño de Contrastación

Para la contrastación de la hipótesis se utilizó el método Pre-Test con el grupo de indicadores. El cual consiste en:

- Una medición previa de la variable dependiente a ser utilizada (Pre-Test)
- La aplicación de la variable independiente a los indicadores de análisis y una nueva medición de la variable dependiente en el grupo de indicadores (Post-Test)

Al finalizar el experimento se establecerán las diferencias entre X_0 y X_1 mostrados en la figura 12, para determinar si hay mejora o no, de acuerdo con estos resultados se darán las conclusiones y recomendaciones pertinentes.

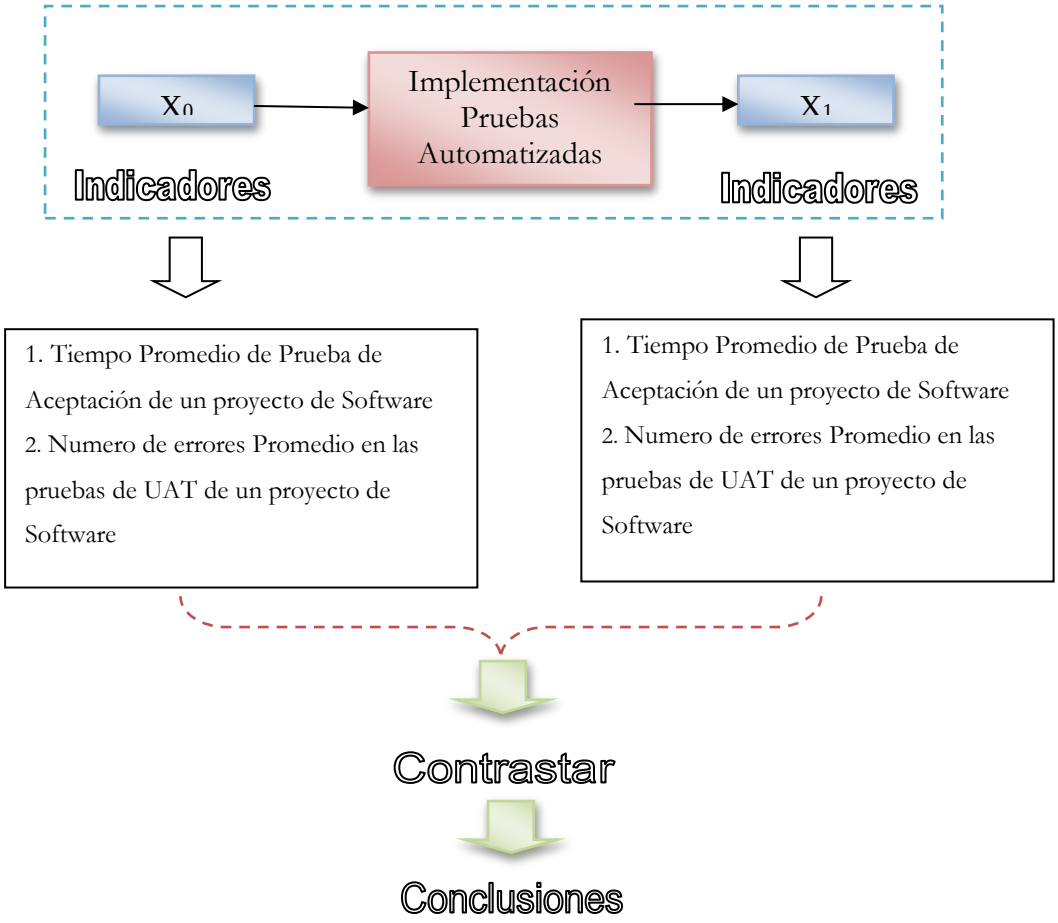


Figura 14: Diseño de Contrastación (Elaboración Propia)

4.4.3 Contrastación de la Hipótesis

Luego de verificar que las muestras se encuentran en una distribución normal, se realizó la contrastación de hipótesis considerando:

a) **Para un indicador n<30** (Prueba t de Student diferencia de medias)

n: muestra ajustada

Nº	Ia	Ip	Di	Di^2
3	I_{3a}	I_{3p}		
4	I_{4a}	I_{4p}		
5	I_{5a}	I_{5p}		
$\frac{\sum_{i=1}^{n} Di}{n}$				$\frac{\sum_{i=1}^{n} \boldsymbol{Di^2}}{\boldsymbol{n}}$

Tabla 7: Cuadro de prueba t de Student

- **Definición de Variables**

Ia = Indicador del Sistema Actual

Ip = Indicador del Sistema Propuesto

- **Hipótesis Estadística**

- Hipótesis Ho=Ia-Ip>=0, Indicador del proceso actual es mejor que el indicador del sistema propuesto.
- Hipótesis H1=Ia – Ip<0, El indicador del sistema propuesto es mejor que el indicador del proceso actual.

- **Nivel de Significancia**

α=5% (error), Nivel de Confianza (1- α)=0.95

- **Estadística de Prueba**

$$T=\frac{D\sqrt{n}}{S_D} \text{…...…….. (1)}$$

Donde:

D = Diferencia de Promedios

n = Muestra

S_D = Desviación Estándar

- **Región de Rechazo (RR)**

La región de rechazo es $t=t^{\alpha}$, donde t^{α} es el valor tabular:

$P[T> t^{\alpha}] = 0.05$, luego RR: $t> t^{\alpha}$

- **Diferencia de Promedios**

$$D =\frac{\sum_{i=1}^{n} D_i}{n} \text{…...…….…….. (2)}$$

- **Conclusiones**

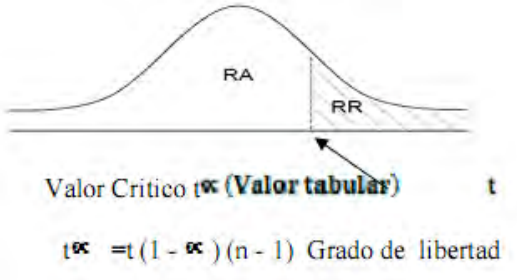


Figura 15: Valor Crítico t de Student

b) Para un indicador n>=30 (Prueba Z diferencia de medias)

n: muestra ajustada

N°	I_a	I_p	$I_{ai} - \overline{I_a}$	$I_{pi} - \overline{I_p}$	$(I_{ai} - \overline{I_a})^2$	$(I_{ai} - \overline{I_p})^2$
1	I_{1a}	I_{1p}				
2	I_{2a}	I_{2p}				
3	I_{3a}	I_{3p}				
..	.	..				
n	I_{na}	I_{np}				
$\sum_{i=1}^{n}(I_{ai} - \overline{I_a})$				$\sum_{i=1}^{n}(I_{pi} - \overline{I_p})$	$\sum_{i=1}^{n}(I_{ai} - \overline{I_a})^2$	$\sum_{i=1}^{n}(I_{pi} - \overline{I_p})^2$

Tabla 8: Cuadro de prueba Z

Ip= Indicador del Sistema Propuesto

- **Hipótesis Estadística**

- Hipótesis Ho=Ia-Ip>=0, Indicador del sistema actual es mejor que el indicador del sistema propuesto.

- Hipótesis H1=Ia – Ip<0, El indicador del sistema propuesto es mejor que el indicador del sistema actual

- **Nivel de Significancia**

α=5% (error), Nivel de Confianza (1- α)=0.95

- **Región de Rechazo (RR)**

La región de rechazo es $z=z^{\alpha}$, donde z^{α} es el valor tabular:

$P[Z> z^{\alpha}] = 0.05$, luego RR: $Z> z^{\alpha}$

- **Diferencia de Promedios**

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- **Desviación Estándar**

$$S^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n \quad 1}$$

$$\sigma_a^2 = \frac{\sum_{i=1}^{n} (v_{ai} \quad T_a)^2}{n \quad 1} \quad ; \quad \sigma_p^2 = \frac{\sum_{i=1}^{n} (v_{pi} \quad T_p)^2}{n \quad 1}$$

- **Conclusión**

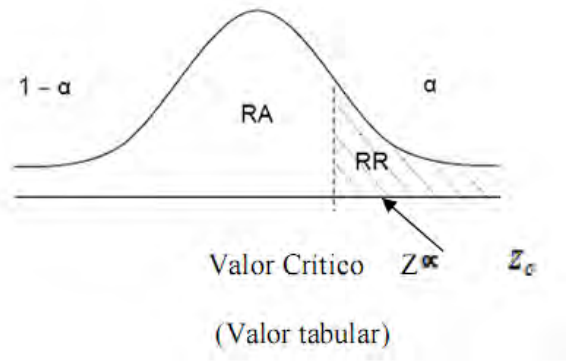


Figura 16: Valor Crítico de prueba Z

4.5 Indicadores

Así mismo para la realización de este diseño se identificaron indicadores cuantitativos donde se evalúan el rendimiento de la ejecución de pruebas manuales y la ejecución de pruebas usando el sistema propuesto.

4.5.1 Métodos de Evaluación

La población y las muestras se calcularán por indicador, los cuales luego se contrastarán, para ello se utilizarán las siguientes fórmulas:

- Fórmula para calcular la muestra:

$$n_1 = \frac{NZ^2PQ}{(N-1)D^2+Z^2PQ} \qquad \text{…...(1)}$$

En (1), donde:

Z = Valor asociado a un Nivel de Confianza (95%) = 1.96

P = Proporción de Éxito = 0.5.

Q = Proporción de Fracaso = 0.5.

D = Error de Muestreo = 0.05.

N = Población

- Fórmula para calcular la muestra ajustada:

$$n = \frac{n_1}{1 + {}^{n_1}/_{N}} \qquad \text{…...…….. (2)}$$

En (2), donde:

n = muestra ajustada.

n_1 = muestra.

N = Población.1.4620

Indicador	Población (N)	Muestra (n_1)	Muestra ajustada (n)	Prueba estadística
I_1	448	207	141	Prueba Z diferencia de medias
I_2	29	29	29	Prueba t Student diferencia de medias

Indicador	Población (N)	Muestra (n_1)	Muestra ajustada (n)	Prueba estadística
I_3	15	15	15	Prueba t Student diferencia de medias
I_4	448	207	141	Prueba Z diferencia de medias
I_5	29	29	29	Prueba t Student diferencia de medias
I_6	15	15	15	Prueba t diferencia de medias

Tabla 9: Cuadro Resumen Población y Muestra

I1. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (Pequeño: 0 – 80 horas).

I2. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (Mediano: 81 – 120 horas).

I3. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (Grande: 121 + horas).

I4. Numero de errores Promedio en las pruebas de UAT en un proyecto de Software. (Pequeño: 0 – 80 horas).

I5. Numero de errores Promedio en las pruebas de UAT en un proyecto de Software. (Mediano: 81 – 120 horas).

I6. Numero de errores Promedio en las pruebas de UAT en un proyecto de software. (Grande: 121 + horas).

Las siguientes condiciones corresponden al uso de las pruebas estadísticas mencionadas anteriormente:

- Si la muestra es menor igual a 30 (n<= 30), entonces se aplica la prueba estadística t student diferencia de medias.
- Si la muestra es mayor a 30 (n > 30), entonces se aplica la prueba estadística Z diferencia de medias.

Indicador	Unidad de Media	Instrumento	Fuente	Informante
$\mathbf{I_1}$ $\mathbf{I_2}$ $\mathbf{I_3}$	Horas	Reporte de Horas por Usuario	Unanet (Sistema de log de horas)	Analista de Calidad de Cignium
$\mathbf{I_4}$ $\mathbf{I_5}$ $\mathbf{I_6}$	Unidad	Lenguaje JQL	Jira	Analista de Calidad de DirecTV

Tabla 10: Cuadro de Indicadores de Validación de Hipótesis

CAPÍTULO V: ANÁLISIS DE RESULTADOS

En el presente capítulo se hará un análisis de los resultados encontrados para todos los indicadores cuantitativos.

5.1 Indicadores Cuantitativos

5.1.1 Tiempo Promedio de Prueba de Aceptación de un proyecto de Software Pequeño

Definición de Variables

T_A: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño usando el proceso manual. (horas)

T_P: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño usando el sistema propuesto (horas).

- **Prueba de Normalidad**

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
TiPreTest	.074	141	.054	.986	141	.181
TiPostTest	.063	141	.200*	.987	141	.230

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Tabla 11: Cuadro de Prueba de Normalidad (SPSS)

Al ser la cantidad de datos mayor a 50, se tomó en cuenta la prueba Kolmogorov-Smirnov.

Variable TiPreTest = T_A:

Sig.: 0.054, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

Variable TiPostTest = T_P:

Sig.: 0.200, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

- **Prueba de Hipótesis**

A. Hipótesis estadísticas

Hipótesis H_0: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño con el proceso manual, es menor que el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño usando el sistema propuesto.

$$H_0 = T_A - T_P <= 0$$

Hipótesis Ha: El Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño con el proceso manual es mayor que el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño usando el sistema propuesto.

$$Ha = T_A - T_P > 0$$

B. Nivel de significancia

Usando un nivel de significancia ($\propto = 0.05$) del 5%. Por lo tanto, el nivel de confianza ($1 - \propto = 0.95$) será del 95%.

C. Estadígrafo de contraste

Puesto que el tamaño de la muestra es n=141, usaremos la distribución normal (Z).

$$\overline{X} = \frac{\sum_{i=1}^{n} Xi}{n}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{X})^2}{n}$$

$$Z_c = \frac{(\overline{X}a - \overline{X}d) - (Xa - Xd)}{\sqrt{(\frac{\sigma_A^2}{n_A} + \frac{\sigma_D^2}{n_D})}}$$

Resultados

Para calcular el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño se ha estimado de una muestra de 141 observaciones. Los datos obtenidos en detalle se muestran en el Anexo Nº 2.

Promedio:

$$\overline{X} = \frac{\sum_{i=1}^{n} Xi}{n}$$

- Promedio de los tiempos usando el proceso Manual:

$$\overline{T_A} = \frac{\sum_{i=1}^{n} T_{Ai}}{n} = \frac{4606}{141} = 32.67$$

- Promedio de los tiempos usando el Sistema Propuesto:

$$\overline{T_P} = \frac{\sum_{i=1}^{n} T_{Pi}}{n} = \frac{3910}{141} = 27.73$$

D. Región Crítica

Con el valor **Zc=7.56394** y para **α =0.05**, debemos interpolar para obtener el valor deseado, usando los valores de la tabla Z. Encontramos

$Z\alpha$ **= 1.645**. Entonces la región critica de la prueba es Zc **= < 1.645,** ∞ **>.**

E. Conclusión:

En el siguiente gráfico podemos observar la Región de aceptación y rechazo para la prueba de la hipótesis Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño.

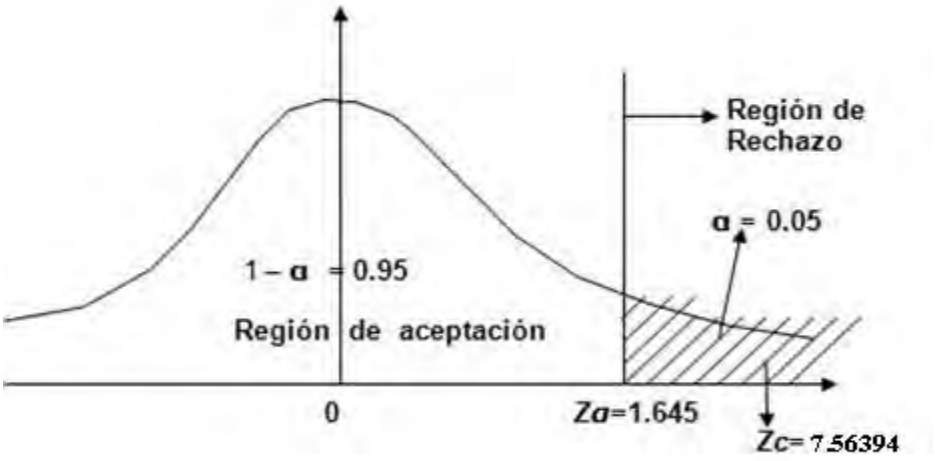


Figura 17: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño

Puesto que Zc **= 7.56394**calculado es mayor que $Z\alpha$ **= 1.645** y estando este valor dentro de la región de rechazo **< 1.645,** ∞ **>,** entonces se rechaza H_0 y por consiguiente se acepta H_a.

Se concluye entonces que el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño es menor con el Sistema propuesto que con el proceso manual actual con un nivel de error del 5% y un nivel de confianza del 95%.

5.1.2 Tiempo Promedio de Prueba de Aceptación de un proyecto de Software Mediano

Definición de Variables

T_A: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano usando el proceso manual. (horas)

T_P: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano usando el sistema propuesto (horas).

- **Prueba de Normalidad**

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
TiPreTest	.165	29	.042	.937	29	.085
TiPostTest	.104	29	.200*	.944	29	.128

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Tabla 12: Cuadro de Prueba de Normalidad (SPSS)

Al ser la cantidad de datos menor a 50, se tomó en cuenta la prueba Shapiro-Wilk.

Variable TiPreTest = T_A:

Sig.: 0.085, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

Variable TiPostTest = T_P:

Sig.: 0.128, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

- **Prueba de Hipótesis**

A) Hipótesis Estadísticas

Hipótesis H_0: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano con el proceso manual, es menor que el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano usando el sistema propuesto.

$$H_0 = T_A - T_P <= 0$$

Hipótesis Ha: El Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano con el proceso manual es mayor que el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano usando el sistema propuesto.

$$Ha = T_A - T_P > 0$$

B) Nivel de significancia

El nivel de significancia (α) escogido para la prueba de la hipótesis es del 5%. Siendo $\alpha =$ 0.05 (nivel de significancia) y 29 -1= 28 grados de libertad, se tiene el valor crítico de T de Student (Ver tabla T de Student en el Anexo N°1):

Valor Crítico: $t_{\alpha=0.05} = -1.833\frac{1}{2}$

Como $\propto$ = 0.05 y n-1 = 29-1 = 28 grados de libertad, la región de rechazo consiste en aquellos valores de t menores que $-t_{0.05}$= -1.833.

C) Resultados de la Hipótesis Estadística

Los datos obtenidos en detalle se muestran en el Anexo N° 3.

a) Diferencia Promedio:

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n}$$

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n} = \frac{-27}{29} = -0.93$$

$$\overline{D} = -0.93$$

b) Desviación Estándar:

$$S_D^2 = \frac{n\sum_{i=1}^{n} D_i^2 - (\sum_{i=1}^{n} D_i)^2}{n(n-1)}$$

$$S_D^2 = \frac{29(75.48) - (-27)^2}{29(28)}$$

$$S_D^2 = 0.28667$$

c) Cálculo de T:

$$S_D^2 = 0.28667$$

$$t = \frac{D\sqrt{n}}{\sqrt{S_D}} = \frac{(-2.7)(\sqrt{29})}{\sqrt{0.28667}}$$

$$t = -15.94686$$

D) Conclusión:

Puesto que: t_c = -15.94686 ($t_{calculado}$) <t_a= -1.833 (t_{abular}), estando este valor dentro de la región de rechazo, se concluye que $V_a - V_p < 0$, se rechaza $\mathbf{H_0}$y $\mathbf{H_a}$ es aceptada, por lo tanto, se prueba la validez de la hipótesis con un nivel de error de 5% ($\propto$ =0.05), siendo la implementación del sistema propuesto una alternativa de solución para el problema de investigación.

En el siguiente gráfico podemos ver la Región de aceptación y rechazo para la prueba de la hipótesis Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano.

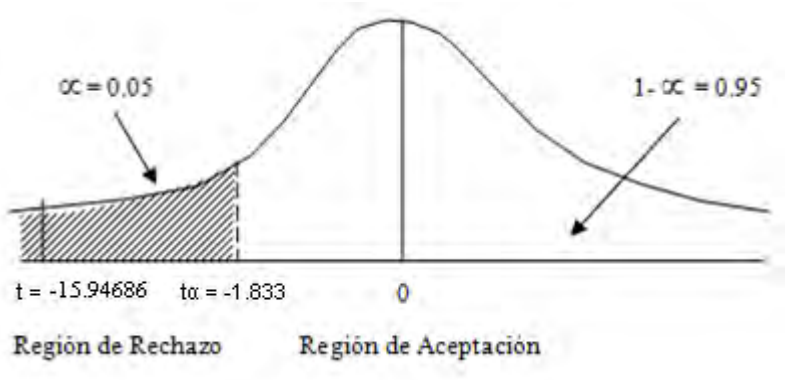


Figura 18: Zona de aceptación y rechazo de Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano

5.1.3 Tiempo Promedio de Prueba de Aceptación de un proyecto de Software Grande

Definición de Variables

T_A: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande usando el proceso manual. (horas)

T_P: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande usando el sistema propuesto (horas).

- **Prueba de Normalidad**

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
TiPreTest	.116	15	.200*	.946	15	.464
TiPostTest	.175	15	.200*	.953	15	.565

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Tabla 13: Cuadro de Prueba de Normalidad (SPSS)

Al ser la cantidad de datos menor a 50, se tomó en cuenta la prueba Shapiro-Wilk.

Variable TiPreTest = T_A:

Sig.: 0.464, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

Variable TiPostTest = T_P:

Sig.: 0.565, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

- **Prueba de hipótesis**

A) Hipótesis Estadísticas

Hipótesis H_0: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande con el proceso manual, es menor que el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande usando el sistema propuesto.

$$H_0 = T_A - T_P <= 0$$

Hipótesis Ha: El Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande con el proceso manual es mayor que el Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande usando el sistema propuesto.

$$Ha = T_A - T_P > 0$$

B) Nivel de significancia

El nivel de significancia (α) escogido para la prueba de la hipótesis es del 5%. Siendo α = 0.05 (nivel de significancia) y 15 -1= 14 grados de libertad, se tiene el valor crítico de T de Student (Ver tabla T de Student en el Anexo N°1):

Valor Crítico: $t_{\alpha=0.05} = -2.713$

Como $\propto$ = 0.05 y n-1 = 15-1 = 14 grados de libertad, la región de rechazo consiste en aquellos valores de t menores que $-t_{0.05} = -2.713$.

C) Resultados de la Hipótesis Estadística

Los datos obtenidos en detalle se muestran en el Anexo 4.

a. Diferencia Promedio:

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n}$$

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n} = \frac{-19}{15} = -1.26$$

$$\overline{D} = -1.26$$

b. Desviación Estándar:

$$S_D^2 = \frac{n\sum_{i=1}^{n} D_i^2 - (\sum_{i=1}^{n} D_i)^2}{n(n-1)}$$

$$S_D^2 = \frac{15(75.48) - (-19)^2}{15(14)}$$

$$S_D^2 = 0.489$$

c. Cálculo de T:

$$S_D^2 = 0.489$$

$$t = \frac{D\sqrt{n}}{\sqrt{S_D}} = \frac{(-2.7)(\sqrt{15})}{\sqrt{0.489}}$$

$$t = -24.468$$

D) Conclusión:

Puesto que: t_c = -24.468 ($t_{calculado}$) <t_a= -2.713 (t_{abular}), estando este valor dentro de la región de rechazo, se concluye que $V_a - V_p < 0$, se rechaza $\mathbf{H_0}$y $\mathbf{H_a}$ es aceptada, por lo tanto, se prueba la validez de la hipótesis con un nivel de error de 5% ($\propto$ =0.05), siendo la implementación del sistema propuesto una alternativa de solución para el problema de investigación.

En el siguiente gráfico podemos ver la Región de aceptación y rechazo para la prueba de la hipótesis Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande.

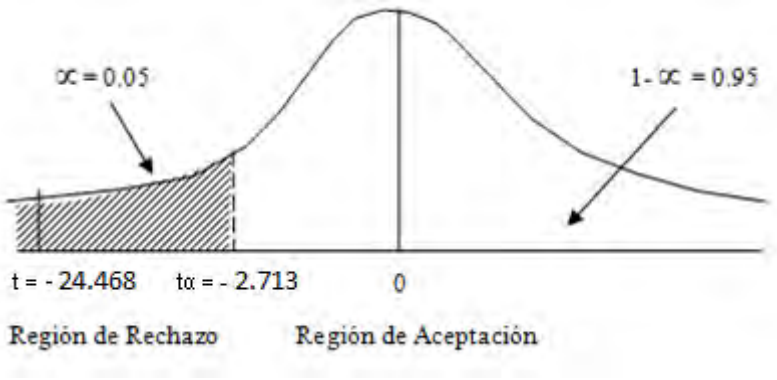


Figura 19: Zona de aceptación y rechazo de Tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande

5.1.4 Número de errores Promedio en las pruebas de UAT para un proyecto de Software Pequeño

Definición de Variables

T_A: Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño usando el proceso manual

T_P: Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño usando el sistema propuesto.

- **Prueba de Normalidad**

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
TiPreTest	.072	141	.069	.991	141	.522
TiPostTest	.067	141	.200*	.981	141	.046

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Tabla 14: Cuadro de Prueba de Normalidad (SPSS)

Al ser la cantidad de datos mayor a 50, se tomó en cuenta la prueba Kolmogorov-Smirnov.

Variable TiPreTest = T_A:

Sig.: 0.069, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

Variable TiPostTest = T_P:

Sig.: 0.200, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

- **Prueba de Hipótesis**

A. Hipótesis estadísticas

Hipótesis H_0: Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño usando el proceso manual es menor que el Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño usando el sistema propuesto.

$$H_0 = T_A - T_P <= 0$$

Hipótesis Ha: Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño usando el proceso manual es mayor que el Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño usando el sistema propuesto.

$$Ha = T_A - T_P > 0$$

B. Nivel de significancia

Usando un nivel de significancia ($\propto$ = 0.05) del 5%. Por lo tanto, el nivel de confianza (1 - $\propto$ = 0.95) será del 95%.

C. Estadígrafo de contraste

Puesto que el tamaño de la muestra es n=141, usaremos la distribución normal (Z).

$$\overline{X} = \frac{\sum_{i=1}^{n} Xi}{n}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{X})^2}{n}$$

$$Z_c = \frac{(\overline{X}a - \overline{X}d) - (Xa - Xd)}{\sqrt{(\frac{\sigma_A^2}{n_A} + \frac{\sigma_D^2}{n_D})}}$$

Resultados

Para calcular el Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño se ha estimado de una muestra de 141 observaciones. Los datos obtenidos en detalle se muestran en el Anexo 5:

Promedio:

$$\overline{X} = \frac{\sum_{i=1}^{n} Xi}{n}$$

- Promedio de los tiempos usando el proceso manual:

$$\overline{T_A} = \frac{\sum_{i=1}^{n} T_{Ai}}{n} = \frac{889}{141} = 6.31$$

- Promedio de los tiempos usando el Sistema Propuesto:

$$\overline{T_P} = \frac{\sum_{i=1}^{n} T_{Pi}}{n} = \frac{726}{141} = 5.15$$

E) Región Crítica

Con el valor **Zc=4.2446** y para **α =0.05**, debemos interpolar para obtener el valor deseado, usando los valores de la tabla Z. Encontramos ***Zα* = 1.143**. Entonces la región critica de la prueba es ***Zc* = < 1.143, ∞ >.**

F) Conclusión:

En el siguiente gráfico podemos observar la Región de aceptación y rechazo para la prueba de la hipótesis Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño.

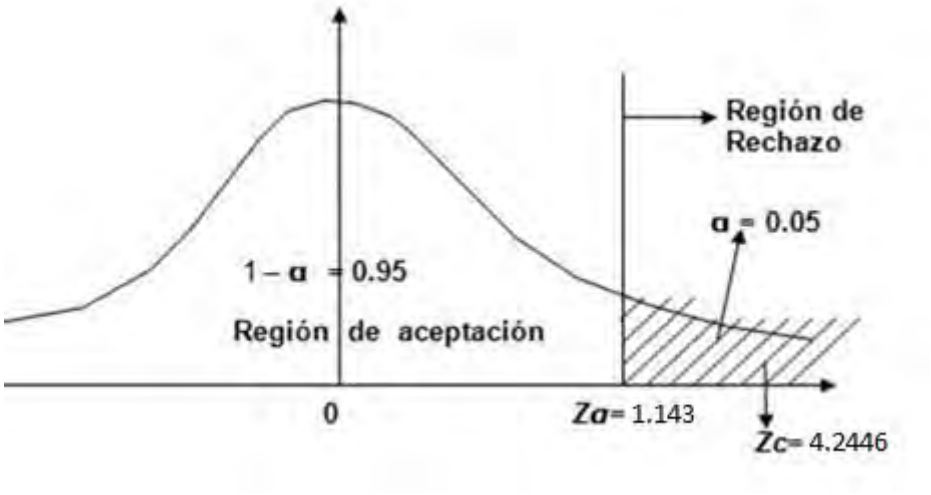


Figura 20: Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño

Puesto que ***Zc* = 4.2446** calculado es mayor que **Zα = 1.143** y estando este valor dentro de la región de rechazo **< 1.143, ∞ >,** entonces se rechaza $\mathbf{H_0}$ y por consiguiente se acepta $\mathbf{H_a}$.

Se concluye entonces que el Número de errores Promedio en las pruebas de UAT para un proyecto de Software pequeño es menor usando el Sistema propuesto que usando el proceso manual con un nivel de error del 5% y un nivel de confianza del 95%.

5.1.5 Número de errores Promedio en las pruebas de UAT para un proyecto de Software Mediano

Definición de Variables

T_A: Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano usando el proceso manual.

T_P: Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano usando el sistema propuesto.

- **Prueba de Normalidad**

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
TiPreTest	.131	29	.200*	.954	29	.228
TiPostTest	.131	29	.200*	.934	29	.068

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Tabla 15: Cuadro de Prueba de Normalidad (SPSS)

Al ser la cantidad de datos menor a 50, se tomó en cuenta la prueba Shapiro-Wilk.

Variable TiPreTest = T_A:
Sig.: 0.228, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

Variable TiPostTest = T_P:
Sig.: 0.068, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

- **Prueba de Hipótesis**

A) Hipótesis Estadísticas

Hipótesis H_0: Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano con el proceso manual, es menor que el Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano usando el sistema propuesto.

$$H_0 = T_A - T_P <= 0$$

Hipótesis Ha: Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano con el proceso manual es mayor que el Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano usando el sistema propuesto.

$$Ha = T_A - T_P > 0$$

B) Nivel de significancia

El nivel de significancia (α) escogido para la prueba de la hipótesis es del 5%. Siendo $\alpha = 0.05$ (nivel de significancia) y n -1= 28 grados de libertad, se tiene el valor crítico de T de Student (Ver tabla T de Student en el Anexo Nº1):

Valor Crítico: $t_{\alpha=0.05} = -1.833\frac{1}{2}$

Como $\propto = 0.05$ y n-1 = 29-1 = 28 grados de libertad, la región de rechazo consiste en aquellos valores de t menores que $-t_{0.05} = -1.833$.

C) Resultados de la Hipótesis Estadística

Los datos obtenidos en detalle se muestran en el Anexo Nº 6.

d) Diferencia Promedio:

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n}$$

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n} = \frac{-46}{29} = -1.58$$

$$\overline{D} = -1.58$$

e) Desviación Estándar:

$$S_D^2 = \frac{n\sum_{i=1}^{n} D_i^2 - (\sum_{i=1}^{n} D_i)^2}{n(n-1)}$$

$$S_D^2 = \frac{29(75.48) - (-46)^2}{29(28)}$$

$$S_D^2 = 0.3668$$

f) Cálculo de T:

$$S_D^2 = 0.3668$$

$$t = \frac{D\sqrt{n}}{\sqrt{S_D}} = \frac{(-1.58)(\sqrt{10})}{\sqrt{0.3668}}$$

$t =$ -10.3545

D) Conclusión:

Puesto que: t_c = -10.3545 ($t_{calculado}$) <t_a= -1.833 (t_{abular}), estando este valor dentro de la región de rechazo, se concluye que $V_a - V_p < 0$, se rechaza $\mathbf{H_0}$y $\mathbf{H_a}$ es aceptada, por lo tanto, se prueba la validez de la hipótesis con un nivel de error de 5% ($\propto$ =0.05), siendo la implementación del sistema propuesto una alternativa de solución para el problema de investigación.

En el siguiente gráfico podemos ver la Región de aceptación y rechazo para la prueba de la hipótesis Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano.

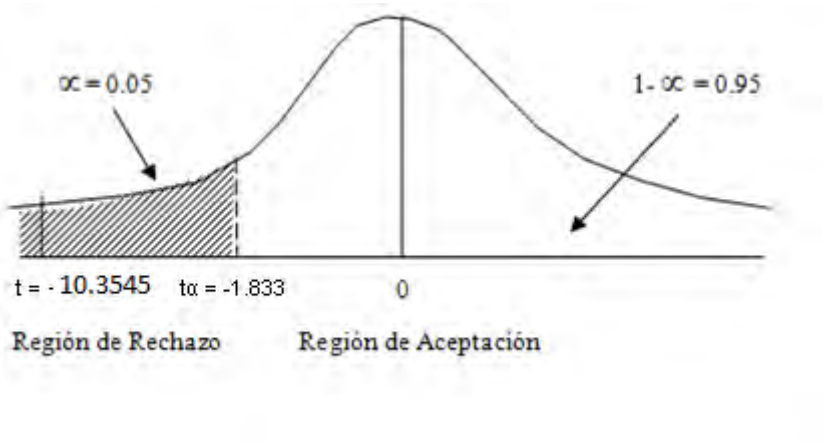


Figura 21: Zona de aceptación y rechazo de Número de errores Promedio en las pruebas de UAT de un proyecto de Software mediano

5.1.6 Número de errores Promedio en las pruebas de UAT para un proyecto de Software Grande

Definición de Variables

T_A: Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande usando el proceso manual.

T_P: Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande usando el sistema propuesto.

- **Prueba de Normalidad**

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
TiPreTest	.141	15	.200*	.892	15	.073
TiPostTest	.205	15	.091	.887	15	.061

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Tabla 16: Cuadro de Prueba de Normalidad (SPSS)

Al ser la cantidad de datos menor a 50, se tomó en cuenta la prueba Shapiro-Wilk.

Variable TiPreTest = T_A:

Sig.: 0.073, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

Variable TiPostTest = T_P:

Sig.: 0.061, al ser mayor de 0.05 se acepta la hipótesis nula, por lo tanto, se puede afirmar que sigue una distribución normal.

- **Prueba de Hipótesis**

A) Hipótesis Estadísticas

Hipótesis H_0: Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande con el proceso manual, es menor que el Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande usando el sistema propuesto.

$$H_0 = T_A - T_P <= 0$$

Hipótesis Ha: Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande con el proceso manual es mayor que el Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande usando el sistema propuesto.

$$Ha = T_A - T_P > 0$$

B) Nivel de significancia

El nivel de significancia (α) escogido para la prueba de la hipótesis es del 5%.
Siendo $\alpha = 0.05$ (nivel de significancia) y n -1= 14 grados de libertad, se tiene el valor crítico de T de Student (Ver tabla T de Student en el Anexo N°1):

Valor Crítico: $t_{\alpha=0.05} = -2.713$

Como $\propto$ = 0.05 y n-1 = 15-1 = 14 grados de libertad, la región de rechazo consiste en aquellos valores de t menores que $-t_{0.05}$= -2.713.

C) Resultados de la Hipótesis Estadística

Los datos obtenidos en detalle se muestran en el Anexo Nº 7.

g) Diferencia Promedio:

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n}$$

$$\overline{D} = \frac{\sum_{i=1}^{n} Di}{n} = \frac{-21}{15} = -1.4$$

$$\overline{D} = -1.4$$

h) Desviación Estándar:

$$S_D^2 = \frac{n\sum_{i=1}^{n} D_i^2 - (\sum_{i=1}^{n} D_i)^2}{n(n-1)}$$

$$S_D^2 = \frac{15(75.48) - (-21)^2}{15(14)}$$

$$S_D^2 = 0.2465$$

i) Cálculo de T:

$$S_D^2 = 0.2465$$

$$t = \frac{D\sqrt{n}}{\sqrt{S_D}} = \frac{(-1.4)(\sqrt{15})}{\sqrt{0.2465}}$$

t = -11.3254

D) Conclusión:

Puesto que: t_c = -11.3254 ($t_{calculado}$) <t_a= -2.713 (t_{abular}), estando este valor dentro de la región de rechazo, se concluye que $V_a - V_p < 0$, se rechaza $\mathbf{H_0}$y $\mathbf{H_a}$ es aceptada, por lo tanto, se prueba la validez de la hipótesis con un nivel de error de 5% ($\propto$ =0.05), siendo la implementación del sistema propuesto una alternativa de solución para el problema de investigación.

En el siguiente gráfico podemos ver la Región de aceptación y rechazo para la prueba de la hipótesis Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande.

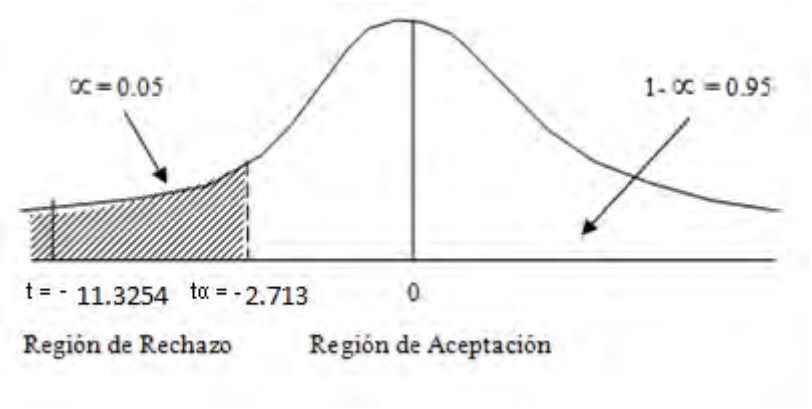


Figura 22: Zona de aceptación y rechazo de Número de errores Promedio en las pruebas de UAT de un proyecto de Software grande

5.2 Discusión de Resultados

I1. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (pequeño: 0 – 80 horas)

Tiempo Promedio de Prueba de Aceptación de un proyecto de Software Pequeño					
Proceso Manual		**Sistema Propuesto**		**Reducción**	
Tiempo (h)	**Porcentaje**	**Tiempo (h)**	**Porcentaje**	**Tiempo (h)**	**Porcentaje**
30.50	100%	27.11	88.91%	3.38	11.09%

Tabla 17: Resultado del Indicador I1

Conclusión: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeño es de 30.50 (100%), usando el sistema se ha disminuido a 27.11 (88.91% del tiempo anterior), teniendo un impacto de mejora de 4.94 (11.09%).

I2. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (mediano: 81 – 120 horas).

Tiempo Promedio de Prueba de Aceptación de un proyecto de Software Mediano					
Proceso Manual		**Sistema Propuesto**		**Reducción**	
Tiempo (h)	**Porcentaje**	**Tiempo (h)**	**Porcentaje**	**Tiempo (h)**	**Porcentaje**
85.24	100%	59.31	69.58%	25.93	30.42%

Tabla 18: Resultado del Indicador I2

Conclusión: El Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediano es de 85.24 (100%), usando el sistema se ha disminuido a 59.31 (69.58% del tiempo anterior), teniendo un impacto de mejora de 25.93 (30.42%).

I3. Tiempo Promedio de Prueba de Aceptación de un proyecto de Software (Grande: 121 + horas).

Tiempo Promedio de Prueba de Aceptación de un proyecto de Software Grande					
Proceso Manual		**Sistema Propuesto**		**Reducción**	
Tiempo (h)	**Porcentaje**	**Tiempo (h)**	**Porcentaje**	**Tiempo (h)**	**Porcentaje**
133.17	100%	80.94	60.77%	52.23	39.23%

Tabla 19: Resultado del Indicador I3

Conclusión: El tiempo Promedio de Prueba de Aceptación de un proyecto de Software grande es de 133.17 (100%), usando el sistema se ha disminuido a 80.94 (60.77% del tiempo anterior), teniendo un impacto de mejora de 52.23 (39.23%).

I4. Número de errores Promedio en las pruebas de UAT. (Pequeño: 0 – 80 horas).

Número de errores Promedio en las pruebas de UAT – Proyecto Software Pequeño					
Proceso Manual		**Sistema Propuesto**		**Reducción**	
Cantidad	**Porcentaje**	**Cantidad**	**Porcentaje**	**Cantidad**	**Porcentaje**
10.66	100%	8.11	76.09%	2.55	23.91%

Tabla 20: Resultado del Indicador I4

Conclusión: El Numero de errores Promedio en las pruebas de UAT **–** Proyecto Software pequeño es de 10.66 (100%), usando el sistema se ha disminuido a 8.11 (76.09% de la cantidad anterior), teniendo un impacto de mejora de 2.55 (23.91%).

I5. Número de errores Promedio en las pruebas de UAT. (Mediano: 81 – 120 horas).

Número de errores Promedio en las pruebas de UAT - Proyecto Software Mediamo					
Proceso Manual		**Sistema Propuesto**		**Reducción**	
Cantidad	**Porcentaje**	**Cantidad**	**Porcentaje**	**Cantidad**	**Porcentaje**
23.4	100%	17.72	75.81%	5.65	24.19%

Tabla 21: Resultado del Indicador I5

Conclusión: El número de errores Promedio en las pruebas de UAT - Proyecto Software Mediano es de 23.4 (100%), usando el sistema se ha disminuido a 17.72 (75.81% de la cantidad anterior), teniendo un impacto de mejora de 5.65 (24.19%).

I6. Número de errores Promedio en las pruebas de UAT. (grande: 121 + horas).

Número de errores Promedio en las pruebas de UAT - Proyecto Software Grande					
Proceso Manual		**Sistema Propuesto**		**Reducción**	
Cantidad	**Porcentaje**	**Cantidad**	**Porcentaje**	**Cantidad**	**Porcentaje**
23.9	100%	17.1	71.54%	6.8	28.46%

Tabla 22: Resultado del Indicador I6

Conclusión: El número de errores Promedio en las pruebas de UAT - Proyecto Software Grande es de 23.9 (100%), usando el sistema se ha disminuido a 17.1 (71.54% del tiempo anterior), teniendo un impacto de mejora de 6.8 (28.46%).

Indicador	Antes		Después		Conclusión
	nA	$\bar{X}A$	nD	$\bar{X}D$	
I1	141	32.67	141	27.73	Se acepta
I2	29	85.74	29	61.94	Se acepta
I3	15	133.17	15	80.94	Se acepta
I4	141	6.31	141	5.15	Se acepta
I5	29	23.4	29	18.56	Se acepta
I6	15	23.9	15	17.1	Se acepta

Tabla 23: Pruebas de Hipótesis de los Indicadores Cuantitativos

CAPÍTULO VI: RESULTADOS

El presente capítulo tiene como finalidad mostrar las conclusiones y recomendaciones después de haber completado la investigación.

6.1 Resumen de Resultados

- El tiempo promedio que lleva realizar las pruebas de aceptación en los 3 tamaños de proyectos de software considerados (pequeño, mediano y grande) se redujo al utilizar el sistema implementado para su automatización.
- El número de errores promedio en las pruebas de UAT en los 3 tamaños de proyectos de software considerados (pequeño, mediano y grande) se redujo al utilizar el sistema implementado para su automatización.
- Cabe mencionar que la reducción en los tiempos de ejecución de las pruebas de aceptación es considerablemente mayor, en cuanto mayor es el tamaño del proyecto. Esto debido a que la posibilidad de ejecución de la misma prueba de aceptación se incrementa cuando el número de defectos encontrados es mayor.

CONCLUSIONES

- Para la implementación del sistema web propuesto, fue de mucha relevancia la participación de los analistas de calidad encargados de realizar las pruebas de aceptación dentro de la empresa. Al final de cada iteración y entrega del sistema con las nuevas funcionalidades a los analistas de calidad, so tomó en cuenta los comentarios y observaciones realizadas.
- A través de la evaluación de los indicadores cuantitativos y usando pruebas estadísticas (prueba t y prueba Z), se logró determinar que el sistema web propuesto reduce el tiempo que se emplea en realizar las pruebas de aceptación. Así también, se logró disminuir la cantidad de errores encontrados en las pruebas de aceptación realizadas por los analistas de calidad del cliente.
- La realización de este proyecto ha logrado automatizar un proceso clave en una de las etapas del desarrollo de software. Logrando reducir los costos en el aseguramiento de calidad del producto de software, como consecuencia de reducir el número de horas hombre invertidas en su realización.
- El marco de trabajo permitió estandarizar las pruebas de aceptación, así como la elaboración de estas.

RECOMENDACIONES

- Se debe fomentar el uso del sistema en los diferentes proyectos de software en los diferentes clientes de la empresa. Actualmente solo se está considerando DirecTV como cliente piloto.
- Se debe evaluar la posibilidad de incluir nuevas funcionalidades en el sistema, que puedan abarcar las necesidades específicas de otros clientes, tales como pruebas en teléfonos móviles y tablets.
- Se debe extender el alcance de las pruebas, agregando las pruebas de integración de los sistemas y servicios, así como pruebas de seguridad.
- Se debe establecer un equipo responsable encargado de dar mantenimiento al sistema, y atender nuevos requerimientos.
- Se debe considerar la posibilidad de integrar las pruebas de carga y estrés con el sistema implementado.

REFERENCIAS BIBLIOGRÁFICAS

[1] Andrea Delgado; "Metodología de desarrollo para aplicación con enfoque SOA"; PEDECIBA Informática; Universidad de la República, Montevideo; 2007.

[2] Alicia Salamon, Patricio Maller; "La Integración Continua Aplicada en el Desarrollo de Software en el Ámbito Científico"; Departamento de Informatica, Instituto Universitario Aeronautico.

[4] Continuous Integration por Martin Fowler, http://martinfowler.com/articles/continuousIntegration.html

[5] Brooks G. Team Pace Keeping Build Times Down. Agile GILE 08 Conference (2008).

[6] Spinellis D. "Software Builders" IEEE Software. Vol 25, 22-23 (2008).

[7] Michael Hutterman; "DevOps for Developers"; Apress; September 2012

[8] Mitesh Soni, "DevOps for Web Development"; Packt Publishing; October 2016

[9] O'Reilly Media, Inc; "DevOps in Practice"; O'Reilly Media, Inc; June 2014

[10] IBM Redbooks; "Using Liberty for DevOps, Continuous Delivery, and Deployment"; IBM Redbooks; November 2015

[11] Paul Swartout; "Continuous Delivery and DevOps"; Packt Publishing; December 2014

[12] Sriram Narayan, "Agile TI Organization Design: For Digital Transformation and Continuous Delivery"; Addison-Wesley Professional, June 2015

[13] Rex Black; "Advanced Software Testing"; Rocky Nook; September 2014

[14] Atlassian Software; https://www.atlassian.com/

[15] Octopus Deploy; https://octopus.com/

[16] Oscar Berta, "Incorporación de la integración Continua en el Desarrollo de Software"; Facultad de ingeniería; Universidad de Piura; 2011.

[17] Mark Fewster, Dorothy Graham; "Experience of Test Automation: Case Studies of Software Test Automation"; Addison-Wesley Professional; January 2012

[18] Grady, R. B. (1999). An economic release decision model: Insights into software project management. In Proceedings of the Applications of Software Measurement Conference, pages 227–239. Orange Park, FL: Software Quality Engineering

[19] Boehm, B. W. (2002). Software engineering economics. SpringerVerlag New York, Inc., New York, NY, USA.

[20] Sharpe, R. (2003). Building a better bug-trap. The Economist. Science Technology Quarterly.

[21] Tassey, G. (2002); "The economic impacts of inadequate infrastructure for software testing. Technical report", National Institute of Standards and Technology

[23] Mike Cohn; "User stories applied: For Agile Software Development"; Addison-Wesley Professional; February 2004

[24] Biolchini, J., et al.; "Systematic review in Software Engineering"; Systems Engineering and Computer Science Department, UFRJ; Rio de Janerio, Brazil; 2005

[25] Michael Kart; "Behavior-driven development: conference tutorial"; St. Edward's University; Austin, TX; April 2012

[27] Tianyao Li, et al; "Gherkin Syntax Extension for Parameterization of Network Switch Configurations in Test Specification"; Software Reliability Engineering Workshops (ISSREW); Toulouse, France; October 2017.

[28] Nan Li, et al; "Skyfire: Model-Based Testing with Cucumber"; Software Testing, Verification and Validation (ICST); Chicago IL, USA; April 2016

[30] Neil Roodyn; "eXtreme .NET: Introducing eXtreme Programming Technique to .NET developers"; Addison-Wesley Professional; December 2004

[31] Cinthia Andres; "Extreme Programming Explained: Embrace Change, Second Edition"; Addison-Wesley Professional; November 2004

[32] Mazedur Rahman, et al; "A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development"; Service-Oriented System Engineering (SOSE); San Francisco Bay CA, USA; April 2015

[33] Ioan Lazar, et al; "Behavior-Driven Development of Foundational UML Components"; Department of Computer Science, Babes-Bolyai University; Romania, August 2010

[34] Azure Active Directory Domain Services - https://azure.microsoft.com/es-es/services/active-directory-ds/

[35] Wayne Ye, "Instant Cucumber BDD How-to"; Pack Publishing; April 2013

[36] John Ferguson, "BDD in Action: Behavior-Driven Development for the whole software lifecycle"; Manning Publications; September 2014

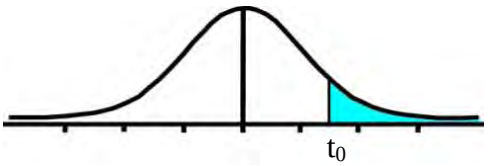
[37] Lasse Koskela, "Test Driven: Practical TDD and Acceptance TDD for Java Developers"; Manning Publications; October 2007

[38] Srincharan Vadapalli, "DevOps: Continuous Delivery, Integration, and Deployment with DevOps"; Packt Publishing; March 2018

ANEXOS

Anexo Nº1

Tabla t-Student



Grados de libertad	0.25	0.1	0.05	0.025	0.01	0.005
1	1.0000	3.0777	6.3137	12.7062	31.8210	63.6559
2	0.8165	1.8856	2.9200	4.3027	6.9645	9.9250
3	0.7649	1.6377	2.3534	3.1824	4.5407	5.8408
4	0.7407	1.5332	2.1318	2.7765	3.7469	4.6041
5	0.7267	1.4759	2.0150	2.5706	3.3649	4.0321
6	0.7176	1.4398	1.9432	2.4469	3.1427	3.7074
7	0.7111	1.4149	1.8946	2.3646	2.9979	3.4995
8	0.7064	1.3968	1.8595	2.3060	2.8965	3.3554
9	0.7027	1.3830	1.8331	2.2622	2.8214	3.2498
10	0.6998	1.3722	1.8125	2.2281	2.7638	3.1693
11	0.6974	1.3634	1.7959	2.2010	2.7181	3.1058
12	0.6955	1.3562	1.7823	2.1788	2.6810	3.0545
13	0.6938	1.3502	1.7709	2.1604	2.6503	3.0123
14	0.6924	1.3450	1.7613	2.1448	2.6245	2.9768
15	0.6912	1.3406	1.7531	2.1315	2.6025	2.9467
16	0.6901	1.3368	1.7459	2.1199	2.5835	2.9208
17	0.6892	1.3334	1.7396	2.1098	2.5669	2.8982
18	0.6884	1.3304	1.7341	2.1009	2.5524	2.8784
19	0.6876	1.3277	1.7291	2.0930	2.5395	2.8609
20	0.6870	1.3253	1.7247	2.0860	2.5280	2.8453
21	0.6864	1.3232	1.7207	2.0796	2.5176	2.8314
22	0.6858	1.3212	1.7171	2.0739	2.5083	2.8188
23	0.6853	1.3195	1.7139	2.0687	2.4999	2.8073
24	0.6848	1.3178	1.7109	2.0639	2.4922	2.7970
25	0.6844	1.3163	1.7081	2.0595	2.4851	2.7874
26	0.6840	1.3150	1.7056	2.0555	2.4786	2.7787
27	0.6837	1.3137	1.7033	2.0518	2.4727	2.7707
28	0.6834	1.3125	1.7011	2.0484	2.4671	2.7633
29	0.6830	1.3114	1.6991	2.0452	2.4620	2.7564
30	0.6828	1.3104	1.6973	2.0423	2.4573	2.7500
31	0.6825	1.3095	1.6955	2.0395	2.4528	2.7440
32	0.6822	1.3086	1.6939	2.0369	2.4487	2.7385
33	0.6820	1.3077	1.6924	2.0345	2.4448	2.7333
34	0.6818	1.3070	1.6909	2.0322	2.4411	2.7284
35	0.6816	1.3062	1.6896	2.0301	2.4377	2.7238

Grados de libertad	0.25	0.1	0.05	0.025	0.01	0.005
36	0.6814	1.3055	1.6883	2.0281	2.4345	2.7195
37	0.6812	1.3049	1.6871	2.0262	2.4314	2.7154
38	0.6810	1.3042	1.6860	2.0244	2.4286	2.7116
39	0.6808	1.3036	1.6849	2.0227	2.4258	2.7079
40	0.6807	1.3031	1.6839	2.0211	2.4233	2.7045
41	0.6805	1.3025	1.6829	2.0195	2.4208	2.7012
42	0.6804	1.3020	1.6820	2.0181	2.4185	2.6981
43	0.6802	1.3016	1.6811	2.0167	2.4163	2.6951
44	0.6801	1.3011	1.6802	2.0154	2.4141	2.6923
45	0.6800	1.3007	1.6794	2.0141	2.4121	2.6896
46	0.6799	1.3002	1.6787	2.0129	2.4102	2.6870
47	0.6797	1.2998	1.6779	2.0117	2.4083	2.6846
48	0.6796	1.2994	1.6772	2.0106	2.4066	2.6822
49	0.6795	1.2991	1.6766	2.0096	2.4049	2.6800

Anexo Nº2

Resultados muestra Pre y Post Test: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software pequeña

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
1	47	25	16.43	-2.33	270.07	5.43
2	22	35	-8.90	8.23	79.27	67.69
3	32	24	1.34	-2.64	1.80	6.96
4	37	34	6.70	7.15	44.82	51.19
5	28	26	-2.19	-0.73	4.79	0.53
6	24	29	-6.11	1.74	37.36	3.01
7	34	29	3.36	2.34	11.29	5.48
8	31	20	0.11	-7.51	0.01	56.45
9	30	30	-0.75	3.34	0.56	11.17
10	32	29	1.78	2.25	3.17	5.06
11	27	22	-3.76	-5.26	14.14	27.70
12	30	26	-0.89	-1.13	0.79	1.27
13	36	25	5.44	-2.30	29.62	5.28
14	35	26	4.72	-1.45	22.25	2.11
15	36	28	6.00	0.92	36.05	0.85
16	19	31	-11.51	3.98	132.46	15.83
17	28	34	-2.92	7.22	8.50	52.09
18	34	30	3.25	3.02	10.55	9.09
19	32	29	1.67	2.04	2.80	4.15
20	36	19	5.47	-8.00	29.95	63.93
21	36	28	5.44	1.06	29.60	1.12
22	35	30	4.37	3.06	19.08	9.34
23	28	35	-2.14	7.94	4.56	63.00
24	28	34	-2.11	6.41	4.46	41.15
25	33	26	2.34	-1.49	5.46	2.21
26	35	31	4.82	3.51	23.26	12.35
27	32	21	1.85	-5.87	3.42	34.46
28	28	23	-2.73	-3.92	7.43	15.39
29	39	24	8.19	-3.12	67.06	9.72
30	29	21	-1.83	-6.08	3.33	36.96
31	28	25	-2.59	-2.59	6.72	6.70
32	33	27	2.90	-0.36	8.41	0.13
33	35	20	4.18	-6.91	17.43	47.78
34	25	28	-5.57	1.25	31.08	1.57
35	39	31	8.43	3.97	71.13	15.73
36	31	21	0.01	-6.24	0.00	38.89
37	24	25	-6.46	-2.44	41.75	5.97
38	31	24	0.58	-3.16	0.34	10.01

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
39	33	36	2.34	8.54	5.48	72.96
40	19	20	-11.25	-7.02	126.59	49.29
41	32	32	1.87	4.46	3.51	19.90
42	21	21	-9.99	-6.59	99.71	43.39
43	26	30	-4.28	3.06	18.32	9.39
44	36	37	5.76	9.75	33.18	95.12
45	30	28	-0.64	0.50	0.41	0.25
46	23	15	-7.34	-12.06	53.93	145.52
47	21	28	-9.58	0.78	91.75	0.60
48	15	25	-15.05	-1.72	226.48	2.94
49	31	28	0.93	1.06	0.87	1.12
50	31	23	0.10	-3.95	0.01	15.58
51	30	28	-0.51	0.46	0.26	0.21
52	32	22	1.67	-5.58	2.78	31.17
53	29	20	-1.25	-7.00	1.57	49.06
54	38	26	7.96	-1.53	63.32	2.34
55	35	27	4.27	-0.56	18.24	0.31
56	34	27	3.62	-0.34	13.07	0.11
57	30	28	-0.91	1.12	0.82	1.25
58	33	27	2.86	-0.23	8.18	0.05
59	27	33	-3.19	5.84	10.16	34.07
60	34	37	3.07	9.44	9.43	89.20
61	22	18	-8.19	-8.80	67.05	77.51
62	32	21	1.64	-5.86	2.68	34.28
63	33	29	2.62	2.04	6.86	4.16
64	33	34	2.66	6.76	7.05	45.69
65	34	33	3.67	5.49	13.48	30.15
66	21	21	-9.57	-6.43	91.67	41.38
67	27	30	-3.56	2.69	12.71	7.23
68	33	30	2.10	2.83	4.40	8.01
69	27	29	-3.92	1.75	15.39	3.05
70	15	30	-15.96	2.54	254.82	6.44
71	36	25	5.58	-1.84	31.12	3.39
72	23	22	-7.86	-5.17	61.70	26.77
73	28	23	-2.87	-4.48	8.23	20.09
74	35	28	4.47	0.50	19.96	0.25
75	25	22	-5.48	-5.55	29.99	30.80
76	33	30	2.23	2.55	4.99	6.50
77	41	29	10.60	1.88	112.27	3.55
78	32	31	1.83	3.99	3.34	15.95
79	25	32	-5.76	4.41	33.23	19.45
80	31	28	0.92	1.18	0.85	1.39

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
81	33	27	2.01	-0.26	4.03	0.07
82	29	27	-1.49	0.31	2.23	0.10
83	39	21	8.32	-6.43	69.29	41.34
84	31	31	0.67	4.28	0.45	18.32
85	31	28	0.28	1.38	0.08	1.90
86	32	38	1.06	11.07	1.13	122.48
87	36	24	5.79	-3.45	33.58	11.94
88	30	37	-0.90	10.33	0.81	106.75
89	32	22	1.04	-4.78	1.08	22.87
90	24	24	-6.63	-2.98	44.00	8.89
91	26	28	-4.57	0.72	20.92	0.52
92	25	26	-5.76	-1.04	33.20	1.08
93	22	24	-8.63	-3.24	74.55	10.49
94	24	20	-6.09	-7.09	37.06	50.28
95	44	38	13.04	10.69	169.99	114.28
96	38	20	7.41	-6.85	54.85	46.89
97	19	31	-11.03	3.59	121.75	12.88
98	33	32	2.52	4.76	6.37	22.61
99	31	15	0.32	-11.81	0.11	139.46
100	29	31	-1.82	3.76	3.30	14.14
101	30	34	-0.17	6.55	0.03	42.93
102	30	34	-0.70	6.66	0.48	44.38
103	28	25	-2.26	-2.37	5.11	5.63
104	30	30	-0.72	2.98	0.52	8.85
105	44	23	13.29	-4.41	176.57	19.44
106	12	26	-18.43	-1.20	339.57	1.45
107	23	33	-7.60	5.90	57.73	34.77
108	28	16	-2.04	-11.19	4.17	125.20
109	39	26	8.65	-1.49	74.87	2.21
110	26	37	-4.34	10.38	18.87	107.70
111	24	28	-6.80	1.15	46.26	1.32
112	44	31	13.03	3.62	169.84	13.07
113	35	27	4.78	-0.31	22.89	0.10
114	30	25	-0.62	-2.07	0.39	4.29
115	26	25	-4.31	-1.67	18.53	2.79
116	28	19	-2.88	-8.16	8.31	66.53
117	27	15	-3.92	-11.73	15.36	137.63
118	27	20	-3.77	-7.52	14.18	56.48
119	32	28	1.72	1.25	2.95	1.56
120	37	26	6.21	-1.60	38.56	2.57
121	33	29	2.61	1.60	6.82	2.57
122	36	28	5.11	0.73	26.13	0.54

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
123	29	26	-1.19	-0.98	1.41	0.96
124	30	22	-0.79	-5.18	0.63	26.85
125	36	29	5.39	1.44	29.10	2.08
126	38	26	7.76	-1.56	60.28	2.44
127	25	25	-5.58	-2.37	31.19	5.63
128	29	30	-1.43	2.58	2.04	6.65
129	37	19	6.46	-7.96	41.72	63.31
130	23	24	-7.77	-2.79	60.34	7.80
131	35	25	4.17	-1.69	17.36	2.85
132	38	32	7.81	5.23	61.01	27.34
133	31	26	0.49	-1.36	0.24	1.84
134	35	28	4.11	1.18	16.90	1.40
135	26	33	-4.19	5.77	17.59	33.31
136	32	37	1.26	9.69	1.58	93.99
137	35	24	4.45	-3.01	19.81	9.04
138	25	35	-5.07	7.68	25.71	59.04
139	35	27	4.34	0.30	18.80	0.09
140	27	32	-3.09	5.11	9.54	26.08
141	36	30	5.97	3.06	35.62	9.38
Suma	4300	3823			4832.48	3606.96
Promedio	32.67	27.63				

Anexo Nº3

Resultados muestra Pre y Post Test: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software mediana

Nº	T_i(Antes)	T_i(Después)
1	81	64
2	83	64
3	80	61
4	84	58
5	81	55
6	88	56
7	83	55
8	85	56
9	85	60
10	83	56
11	82	60
12	86	57
13	86	55
14	90	63
15	81	60
16	85	58
17	88	65
18	88	60
19	88	56
20	85	65
21	88	58
22	86	60
23	82	59
24	90	61
25	86	59
26	90	59
27	82	61
28	88	62
29	88	57
Suma	2472	1720
Promedio	85.74	59.31

Anexo Nº4

Resultados muestra Pre y Post Test: Tiempo Promedio de Prueba de Aceptación de un proyecto de Software Grande

Nº	T_i(Antes)	T_i(Después)
1	140	83
2	138	76
3	135	71
4	130	77
5	129	80
6	134	74
7	129	74
8	120	80
9	127	81
10	140	79
11	124	72
12	136	74
13	131	70
14	122	86
15	138	72
Suma	1973	1149
Promedio	133.17	60.94

Anexo Nº5

Resultados muestra Pre y Post Test: Numero de errores Promedio en las pruebas de UAT en proyectos de Software Pequeños

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
1	7	6	7.23	5.73	52.22	32.84
2	7	12	7.39	11.92	54.67	142.20
3	3	3	3.19	2.77	10.16	7.68
4	15	12	15.39	12.06	236.99	145.38
5	17	6	17.46	6.25	304.83	39.11
6	11	1	10.64	1.17	113.16	1.37
7	10	1	9.72	1.00	94.55	1.00
8	14	14	14.24	13.56	202.72	183.95
9	14	12	13.74	11.56	188.68	133.69
10	12	8	11.51	7.74	132.37	59.85
11	12	9	11.54	8.53	133.27	72.72
12	2	8	2.39	7.97	5.69	63.49
13	9	10	8.61	9.94	74.06	98.84
14	13	11	12.73	11.18	162.00	125.01
15	5	9	5.29	8.65	28.02	74.87
16	14	8	14.09	7.51	198.48	56.47
17	7	7	7.29	6.87	53.10	47.19
18	14	5	13.54	5.32	183.22	28.31
19	11	9	11.32	8.64	128.25	74.72
20	8	12	8.25	11.99	68.13	143.83
21	18	6	17.91	5.58	320.77	31.13
22	14	1	14.08	1.26	198.30	1.59
23	17	2	16.63	2.02	276.47	4.09
24	9	7	8.72	6.74	76.04	45.46
25	14	13	13.95	13.21	194.56	174.56
26	13	7	12.87	6.52	165.68	42.55
27	5	8	5.48	7.67	30.03	58.89
28	16	6	15.97	6.14	254.97	37.68
29	11	10	11.10	10.23	123.32	104.67
30	11	11	11.01	11.32	121.26	128.12
31	5	13	5.49	12.67	30.14	160.50
32	17	9	17.31	9.11	299.48	83.01
33	9	3	8.61	2.51	74.16	6.32
34	7	10	7.18	10.24	51.58	104.85
35	9	12	8.55	12.23	73.06	149.68
36	12	8	12.34	7.66	152.18	58.74
37	9	5	9.07	5.43	82.28	29.50

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
38	9	10	9.32	9.71	86.95	94.23
39	10	3	10.10	3.30	102.01	10.92
40	5	8	4.57	8.35	20.88	69.69
41	16	13	16.40	12.65	269.04	159.91
42	12	16	11.72	16.40	137.36	269.06
43	8	8	7.63	8.04	58.23	64.59
44	16	8	15.59	8.04	243.16	64.67
45	10	6	10.00	6.20	100.06	38.45
46	10	7	9.53	7.04	90.78	49.54
47	8	10	8.22	10.47	67.65	109.65
48	7	7	7.01	7.19	49.07	51.64
49	17	5	17.04	5.44	290.38	29.63
50	7	9	6.75	9.15	45.52	83.70
51	13	6	13.27	5.67	176.04	32.18
52	11	14	10.83	14.29	117.34	204.24
53	3	10	2.53	10.17	6.42	103.37
54	20	8	19.58	7.72	383.23	59.64
55	10	5	10.25	4.54	105.16	20.62
56	11	10	11.39	9.66	129.65	93.39
57	19	15	18.53	15.07	343.18	227.06
58	12	15	12.49	14.78	156.02	218.40
59	5	12	5.28	11.61	27.86	134.87
60	1	4	1.14	4.31	1.30	18.54
61	14	5	13.85	5.21	191.77	27.15
62	15	16	15.46	16.32	239.10	266.40
63	12	3	12.26	3.35	150.26	11.20
64	4	12	3.85	12.12	14.81	146.90
65	9	9	8.93	9.32	79.76	86.86
66	15	4	15.22	3.84	231.68	14.72
67	11	2	10.98	1.68	120.54	2.81
68	15	7	14.93	6.71	222.82	45.08
69	14	3	13.57	2.91	184.26	8.49
70	15	13	14.83	12.88	219.99	165.90
71	13	8	12.95	7.94	167.66	62.99
72	8	11	8.05	10.88	64.75	118.44
73	13	11	12.82	11.31	164.38	128.01
74	10	15	10.18	15.00	103.53	224.88
75	10	12	10.13	12.30	102.59	151.34
76	9	3	9.29	3.13	86.32	9.82
77	8	5	7.84	5.04	61.39	25.37
78	13	5	12.85	4.87	165.17	23.70
79	4	7	4.10	7.05	16.82	49.69

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
80	9	8	9.47	8.34	89.61	69.64
81	9	5	8.81	4.86	77.54	23.66
82	22	9	22.10	9.08	488.38	82.42
83	4	15	3.52	14.62	12.36	213.69
84	14	12	14.24	12.15	202.80	147.60
85	5	12	5.25	12.03	27.57	144.74
86	16	13	15.77	12.89	248.73	166.07
87	16	7	15.90	6.96	252.81	48.40
88	8	11	7.67	10.60	58.84	112.26
89	14	7	13.69	7.11	187.38	50.60
90	8	5	7.76	4.65	60.18	21.63
91	21	6	21.36	6.08	456.16	36.99
92	15	11	14.61	10.89	213.45	118.68
93	7	6	7.23	5.68	52.24	32.25
94	11	3	11.21	2.71	125.60	7.36
95	2	6	2.42	6.04	5.86	36.47
96	16	1	16.28	1.00	264.91	1.00
97	12	9	11.59	8.83	134.25	78.00
98	12	14	12.27	13.92	150.57	193.72
99	8	9	7.51	9.15	56.34	83.73
100	8	7	7.50	6.59	56.28	43.44
101	15	1	14.98	1.03	224.26	1.07
102	9	6	9.03	6.36	81.58	40.40
103	8	7	8.42	6.84	70.96	46.84
104	11	10	10.83	10.28	117.38	105.62
105	10	1	9.99	1.00	99.85	1.00
106	5	8	5.07	8.22	25.70	67.54
107	12	12	11.63	12.40	135.21	153.87
108	6	4	6.28	3.52	39.42	12.39
109	9	13	9.05	12.62	81.88	159.15
110	9	7	8.66	7.28	74.99	53.05
111	15	7	14.76	7.29	217.96	53.11
112	7	8	7.47	8.32	55.82	69.15
113	10	8	10.24	7.70	104.96	59.32
114	9	9	9.50	9.11	90.17	82.90
115	4	9	4.32	8.73	18.66	76.16
116	13	2	12.75	1.76	162.52	3.08
117	11	9	10.53	8.74	110.79	76.45
118	9	7	8.73	7.40	76.20	54.70
119	17	13	16.57	12.61	274.59	159.05
120	19	9	18.76	8.64	351.83	74.68
121	9	8	8.83	8.46	77.95	71.52

Nº	T_i(Antes)	T_i(Después)	X_i-X_{med} (Antes)	X_i-X_{med} (Después)	$(X_i-X_{med})^2$ (Antes)	$(X_i-X_{med})^2$ (Después)
122	8	10	7.83	9.86	61.26	97.32
123	16	5	16.35	5.13	267.24	26.29
124	13	10	13.20	9.83	174.23	96.63
125	1	10	1.38	10.13	1.91	102.71
126	12	8	12.01	8.05	144.29	64.83
127	10	3	9.97	2.68	99.38	7.18
128	11	13	11.33	12.82	128.35	164.31
129	12	10	12.33	10.49	152.09	109.98
130	7	3	7.40	2.86	54.79	8.19
131	9	4	8.90	4.11	79.29	16.88
132	7	13	7.26	12.95	52.68	167.59
133	6	9	6.05	9.24	36.62	85.44
134	7	8	7.48	7.98	55.91	63.69
135	6	9	5.88	9.36	34.56	87.56
136	11	9	10.85	8.96	117.70	80.24
137	10	5	9.68	5.17	93.69	26.69
138	11	9	11.29	9.42	127.56	88.83
139	15	4	14.81	3.70	219.40	13.71
140	13	11	12.65	10.89	160.02	118.66
141	11	5	11.18	5.23	124.91	27.30
Suma	1503	1144			18444.16	11119.01
Promedio	6.31	5.15				

Anexo Nº6

Resultados muestra Pre y Post Test: Numero de errores Promedio en las pruebas de UAT en proyectos de Software Medianos

Nº	T_i(Antes)	T_i(Después)
1	25	19
2	23	19
3	25	15
4	21	17
5	22	18
6	24	17
7	24	17
8	20	21
9	25	20
10	25	20
11	21	21
12	23	19
13	25	18
14	22	21
15	20	17
16	23	15
17	24	16
18	24	16
19	25	15
20	27	18
21	23	15
22	20	18
23	23	16
24	22	16
25	27	17
26	26	16
27	26	20
28	22	19
29	21	18
Suma	678	514
Promedio	23.4	18.56

Anexo Nº7

Resultados muestra Pre y Post Test: Numero de errores Promedio en las pruebas de UAT en proyectos de Software Grandes

Nº	T_i(Antes)	T_i(Después)
1	24	15
2	20	17
3	26	19
4	20	19
5	24	19
6	23	17
7	26	20
8	21	20
9	22	19
10	21	16
11	20	17
12	25	18
13	25	17
14	26	20
15	22	20
Suma	345	273
Promedio	23.9	17.1

Anexo Nº8

Lenguaje JQL para obtención de Población en Jira

Filtros:

Fecha: Año 2016

Proyectos: Solo Proyectos de DirecTV (código: WDTV)

Tamaños existentes de proyectos:

- Tiny (0-12 Hours)
- Small (13-40 Hours)
- Medium (41-80 Hours)
- Large (81-120 Hours)
- X-Large (121-240 Hours)
- XX-Large (241 Hours and Above)

1. **Proyectos pequeños:**

   project = WDTV AND issuetype = "Client Initiative 2" AND labels = manual AND "Size" in ("Medium (41-80 Hours)", "Small (13-40 Hours)", "Tiny (0-12 Hours)") AND "Release Date" >= 2016-1-1 AND "Release Date" <= 2016-12-31

2. **Proyectos Medianos**

   project = WDTV AND issuetype = "Client Initiative 2" AND labels = manual AND "Size" = "Large (81-120 Hours)" AND "Release Date" >= 2016-1-1 AND "Release Date" <= 2016-12-31

3. **Proyectos Grandes**

   project = WDTV AND issuetype = "Client Initiative 2" AND labels = manual AND "Size" in ("X-Large (121-240 Hours)", "XX-Large (241 Hours and Above)") AND "Release Date" >= 2016-1-1 AND "Release Date" <= 2016-12-31

Anexo Nº9

Marco de Trabajo para la elaboración de las pruebas de aceptación

1. Test Case Manager

Se utiliza JIRA como el manejador de casos de pruebas. Toda la información requerida será registrada de dicho sistema.

2. Test Plans

Cada proyecto (Client Initiative 2) tendrá vinculado los test plans (planes de prueba) que abarquen la prueba de la funcionalidad deseada descrita en los requerimientos asociados al proyecto.

Todos los planes de prueba serán visualizados por el sistema web de automatización. El link considerado: **"Is tested by"**

Cada Test Plan contara con una lista de test cases (casos de prueba), los cuales serán ingresados a manera de subtasks.

2.1 Test Cases

Los casos de prueba se escribirán en lenguaje Gherkin, el cual define la estructura y una sintaxis básica para la descripción de las pruebas que pueden ser entendidas tanto por los integrantes técnicos del equipo, así como también por los analistas de calidad.

Elementos requeridos: Feature y Scenario.

Ejemplo:

```
Feature: Some terse yet descriptive text of what is desired
  Textual description of the business value of this feature
  Business rules that govern the scope of the feature
  Any additional information that will make the feature easier to understand

  Scenario: Some determinable business situation
    Given some precondition
      And some other precondition
    When some action by the actor
      And some other action
      And yet another action
    Then some testable outcome is achieved
      And something else we can check happens too

  Scenario: A different situation
      ...
```

- **Feature:** nombre de la funcionalidad que vamos a probar, el título de la prueba.
- **Scenario:** habrá uno por cada prueba que se quiera especificar para esta funcionalidad.
- **Given:** acá se marca el contexto, las precondiciones.
- **When:** se especifican las acciones que se van a ejecutar.
- **Then:** acá se especifica el resultado esperado, las validaciones a realizar.

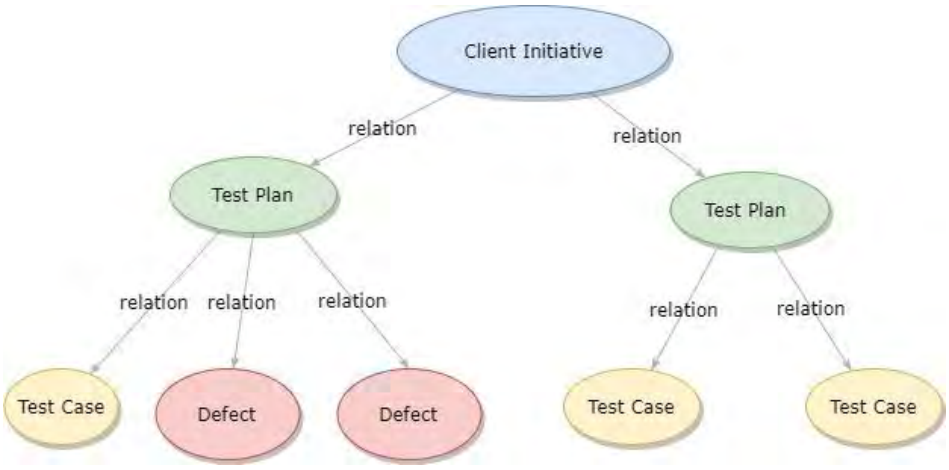
3. Reporte de Errores

Cada error encontrado durante las pruebas de aceptación será reportado como "Defect 2", y será vinculado al "Test Plan".

El Link considerado: **"Is blocked by"**

4. Artefactos Considerados

Los 4 artefactos considerados en la prueba de aceptación, y su relación se muestra en el siguiente gráfico:



- Cada "Client Initiative" puede tener 1 a más "Test Plan"
- Cada "Test Plan" puede tener 1 a más "Test Case"
- Cada "Test Plan" puede tener 1 a más "Defect"

5. Consideraciones

- Todos los cambios realizados a los casos de prueba (código gherkin) deberán ser realizados a través del sistema web de automatización, el cual se encuentra vinculado directamente al sistema JIRA.
- Los defectos podrán ser reportados desde ambos sistemas: JIRA y sistema web de automatización.