UNIVERSIDAD PERUANA DE CIENCIAS E INFORMÁTICA

FACULTAD DE CIENCIAS E INGENIERÍA

INGENIERÍA DE SISTEMAS E INFORMÁTICA

TESIS:

"IMPLEMENTACIÓN DE LA HERRAMIENTA BADBOY PARA LA MEJORA DE PRUEBAS FUNCIONALES EN APLICACIONES WEB"

PRESENTADO POR:

BACH.: GUTIERREZ ZAPATA KAROLY GUADALUPE

PARA OPTAR AL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS E INFORMÁTICA

ASESOR:

MG. SAITO SILVA, CARLOS AGUSTÍN

LIMA – PERÚ
2019

Dedicatoria

A mis padres por su apoyo constante en el logro de mis metas.
A mi hermana por todos esos consejos que me impulsan a ser cada vez mejor.

Agradecimientos

Agradezco a Dios por permitirme llegar hasta donde he llegado.

A mis padres y hermana por su apoyo incondicional en realizarme como persona y profesional.

Asimismo, agradezco a esas personas que forman parte de mí quienes estuvieron junto a mí apoyándome en esos momentos difíciles y por quienes guardo una profunda gratitud. Asimismo, quiero agradecer a esa persona que existe en mis recuerdos por enseñarme el valor del esfuerzo y la perseverancia.
A todos ellos muchas gracias.

Reconocimiento

A mi hermana quien me apoyó en la realización de este proyecto brindándome su tiempo y ofreciéndome parte de sus conocimientos.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

La presente investigación tiene como finalidad presentar la automatización de pruebas funcionales en una consultora de desarrollo de software.

Realizando el análisis del proceso antes de la implementación, se determinó que el principal problema radicaba en el tiempo que se tomaba en realizar las pruebas funcionales. Siendo así se propuso el tema de automatizar las pruebas dentro de la consultora de software.

Para aplicar la automatización se utilizó la herramienta Badboy por lo que se procedió a realizar la automatización de algunos test para que se puedan ejecutar en diferentes situaciones sin supervisión de algún recurso. Al final de cada test la herramienta presenta informes de las secuencias realizadas. En caso de que algunos de los pasos no se hayan ejecutado de manera correcta se guarda en un repositorio una imagen del error.

Luego de la implementación de la herramienta Badboy, se redujo el tiempo en realizar pruebas funcionales por lo que se seguirá implementando en los demás módulos. Con la ayuda del analista funcional, se irá añadiendo nuevos test al script grabado en la herramienta. Por lo que se espera que con esta actividad se encuentren los errores o defectos de software en etapas de pruebas y se reduzcan de manera significativa en los ambientes de producción cada vez que se realice una actualización en la aplicación.

Palabras clave: Pruebas funcionales. Automatización

ABSTRACT

The present investigation have as finality to present the Automation of Functional Test in a software development consultancy.

Doing the analysis of the process before implementation, it was determined that the main problem was in the time taken to perform the Functional test. That being the case, was proposed the Automation of Functional test in the software development consultancy.

To apply automation the badboy tool was used, so the automation of some test was done so that they could be executed in different situations without supervision of any resource. At the end of each test the tool presents reports of the sequences carried out. In case some of the steps have not been executed correctly, an error image is saved in a repository.

After the implementation of badboy tool, the time in perfoming functional test was reduced so it will continue to be implemented in the other modules. With the help of the functional analyst, new tests will be added to the script recorded in the tool. So it is expected that this activity will find errors or software defects in testing stages and will be significantly reduced in the production environments each time an update is made in the application.

Keywords: Functional Testing, Automation

INTRODUCCIÓN

En los últimos años y con el Auge de la tecnología, muchas empresas están a la vanguardia de optimizar y/o mejorar sus procesos, ante esta situación, nacen las consultoras de software que cada día están en constante mejora distinguiéndose de la competencia por ofrecer al cliente productos de calidad en el menor tiempo.

La presente tesis titulada: "IMPLEMENTACIÓN DE LA HERRAMIENTA BADBOY PARA LA MEJORA DE PRUEBAS FUNCIONALES EN APLICACIONES WEB", se basa en la puesta en marcha de la herramienta mencionada en una consultora de software con la finalidad de reducir el tiempo en realizar las pruebas funcionales y mejorar la calidad del software. Por lo que mediante esta implementación la imagen de la empresa mejoró frente a sus clientes debido a la entrega de los requerimientos en el tiempo establecido con las respectivas validaciones ya que con esta investigación se logró reducir el tiempo en realizar las respectivas pruebas además de la reducción en forma considerable los fallos o errores en el software gracias a que se identificaron en el entorno de desarrollo.

El presente proyecto de tesis consta de cuatro capítulos:
En el primer capítulo, se describe la problemática actual que presenta la consultora de software; también se define los objetivos de investigación; la justificación teórica, metodológica, práctica y social; importancia y limitaciones de la investigación

En el segundo capítulo, se detallan los antecedentes de la presente investigación en los que se presentan trabajos que han sido realizados por otras personas y que sirvieron de referencia para la elaboración de esta tesis; también se desarrolló los distintos temas relacionados al presente proyecto; asimismo se describe los términos básicos que se utilizaron en la tesis, así como la hipótesis, variables e indicadores de la variable dependiente.

En el tercer capítulo, se presenta el diseño metodológico en la que se considera el diseño, tipo, nivel, enfoque de investigación, la población y muestra. Así como las técnicas e

instrumentos de recolección de datos, técnicas para el análisis de datos que se utilizaron para la presente investigación.

En el cuarto capítulo, se realiza la presentación de resultados pre-test y post-test de las variables estudiadas incluyendo su respectivo análisis que determinó que el presente proyecto es viable.

Finalmente se incluye las conclusiones de las hipótesis y las recomendaciones para que este proyecto siga teniendo los mismos o mejores resultados.

CAPÍTULO I: PLANEAMIENTO DEL ESTUDIO

1.1 Descripción del Problema de Investigación

En los últimos años, han surgido diversas empresas dedicadas al desarrollo de software, ello debido a que otras grandes y/o pequeñas empresas de otro giro de negocio tales como el sector bancario, sector minero, sector pesquero entre otras empresas requieren automatizar sus procesos utilizando sistemas de información

Muchas de estas empresas de desarrollo de Software se ven en la necesidad de entregar a sus clientes los requerimientos solicitados en el menos tiempo posible, ello debido a la competencia.

Sin embargo, muchas veces el entregable no pasa por las validaciones necesarias lo cual, al realizar el despliegue en producción ocurren diversos fallos en la aplicación. Dado que muchas de estas consultoras no cuentan con un área de QA o sus procesos de validación son manuales.

La empresa seleccionada para la implementación, es una organización que desarrolla soluciones tecnológicas innovadoras de acuerdo a las necesidades de sus clientes. (Ver Figura 01)

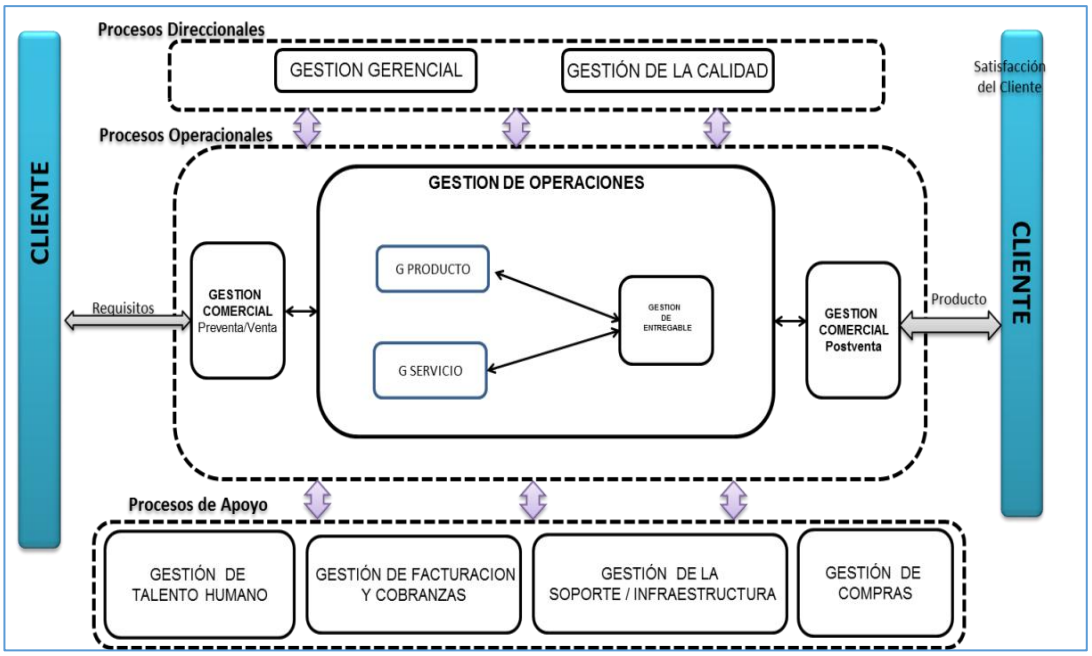

Figura 01: Mapa de procesos de consultora de software
Elaboración: propia

Esta consultora además de realizar desarrollos a medida cuenta con un producto estándar desarrollado el cual es una aplicación web de Recursos humanos que tiene múltiples funcionalidades. Diversas empresas peruanas cuentan con esta aplicación instalada en sus ambientes y a las que periódicamente se realizan actualizaciones por los siguientes motivos:

✓ Nuevos Requerimientos, los clientes solicitan mejoras a la aplicación de acuerdo a las necesidades del negocio

✓ Corrección de un error en el software, el cual puede ser causado por la actualización de un nuevo requerimiento o por la regularización de actualización.

✓ Regularización de actualización, esto puede ser por solicitud del cliente o por proposición misma de la consultora para que la aplicación tenga nuevas mejoras desarrolladas.

Para el caso de Nuevo Requerimiento, el proceso es el siguiente:
Las mejoras que solicitan los clientes son llamados requerimientos los cuales son canalizados en reuniones con el jefe de proyecto (JP), el cual detalla el Requerimiento (RQ) en un documento denominado *Documento de Especificación de Requerimientos.*

El JP envía el RQ mediante correo electrónico al Analista funcional (AF) el cual a su vez estima el tiempo de desarrollo del RQ y elabora un documento denominado *Documento de Especificación Técnica.*

En este documento el AF detalla los objetos de Base de Datos (BD) y/o páginas aspx que se crearán y/o modificarán (Ver Figura 02).

El Programador una vez terminado el desarrollo de RQ, realiza las pruebas unitarias.

Una vez terminada dicha validación envía los objetos de BD y páginas aspx creados y/o modificados al Release Manager (RM), quien se encarga de realizar el versionado de objetos de BD y paginas aspx y el posterior despliegue en el ambiente de Desarrollo.

Una vez que el ambiente de Desarrollo se encuentra actualizado el Release Manager notifica mediante correo electrónico al desarrollador y al AF para la validación del RQ.

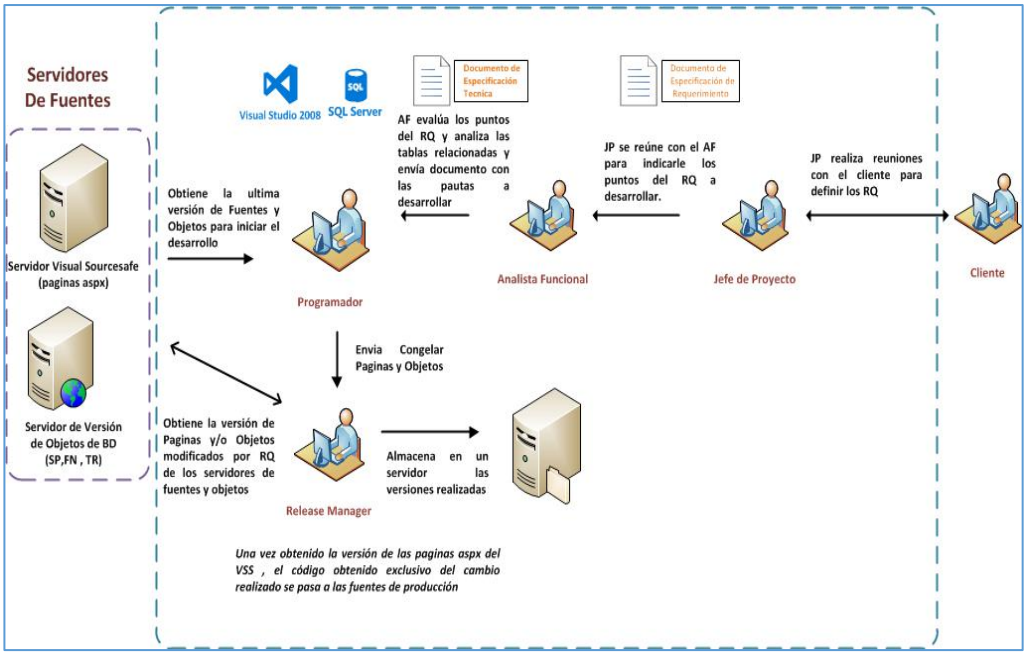

Figura 02: Proceso actual del versionado de objetos
Elaboración: propia

Muchas veces no se llegan a finalizar todas las pruebas respectivas debido a la falta de tiempo ocurriendo diversos fallos en el sistema una vez aplicado la actualización lo que genera incomodidad y malestar por parte del cliente. (Ver Figura 03)

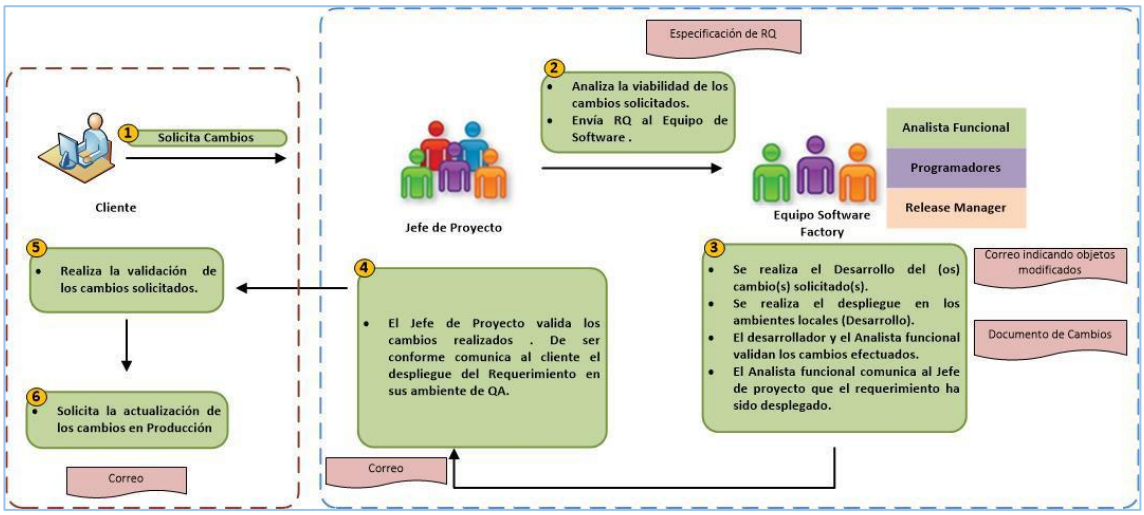

Figura 03: Proceso actual de despliegue de versiones
Elaboración: propia

Cuando es el caso de corregir un error en la aplicación, el área de centro de atención al cliente (CAC) previo análisis para verificar si es error a causa de un mal desarrollo deriva al área de Software Factory donde el Analista funcional designa a un desarrollador para la corrección del mismo. Una vez terminado la corrección envía al release manager por correo los objetos a congelar para que posteriormente el release manager realice el despliegue en el ambiente de desarrollo seguidamente este notifica mediante un correo al analista funcional y al desarrollador. Luego de ello el desarrollador juntamente al analista realizan la validación respectiva para posteriormente programar el pase a producción.

Para el caso de regularización de actualización, el área de centro de atención al cliente coordina directamente con el cliente sobre el horario en que se realizará la actividad. Previamente a ello se ha actualizado el ambiente de desarrollo. Sin embargo, muchas veces como se comentó anteriormente no se llega a validar toda la aplicación debido al tiempo que demanda validar cada una de las funcionalidades.

La presente tesis, investiga la manera de implementar una herramienta de automatización de pruebas funcionales para poder entregar al cliente un producto en el tiemplo establecido, con valor y así evitar interrumpir la continuidad del negocio.

1.2 Delimitación del Problema de Investigación

1.2.1 Espacial

El proyecto se encuentra delimitado en una consultora de desarrollo de software, su aplicación será bajo el entorno de una aplicación web de Recursos Humanos.

1.2.2 Temporal

El presente proyecto se realiza en los meses indicados:

Proyecto de Tesis

- Inicio: Enero 2018
- Fin: Febrero 2018

Borrador de Tesis

- Inicio: Marzo 2018
- Fin: Mayo 2018

1.3 Formulación del Problema de Investigación

1.3.1 Problema general

¿Cómo mejorar las pruebas funcionales en aplicaciones web?

1.3.2 Problemas Específicos

a) ¿Cómo identificar fallos en el desarrollo de Software?

b) ¿Cómo reducir el tiempo en las pruebas funcionales básicas de las reglas de negocio?

c) ¿Cómo reducir los costos en corregir una falla en el software?

1.4 Planteamiento de los Objetivos de la Investigación

1.4.1 Objetivo General

Implementar la herramienta Badboy, para mejorar el entorno de pruebas funcionales en aplicaciones web.

1.4.2 Objetivos Específicos

a) Describir el método de pruebas funcionales del analista funcional, para identificar los fallos en el desarrollo de software.

b) Automatizar las pruebas funcionales de aplicaciones web para reducir el tiempo en las pruebas funcionales básicas de las reglas de negocio.

c) Rediseñar el proceso de trabajo entre el desarrollo y las pruebas funcionales de software para reducir los costos en solucionar fallas en el software.

1.5 Justificación e Importancia de la Investigación

1.5.1 Justificación

Justificación Teórica

Para el presente proyecto se considera la implementación del software Badboy ya que es una herramienta compatible con el navegador Internet Explorer y todos los productos desarrollados por Microsoft. Es por ello, debido a las características de la aplicación web, Badboy es la herramienta que se ajusta más al entorno web donde se quiere implementar.

Justificación Metodológica

Para el desarrollo del presente proyecto, implementar la herramienta Badboy, es de suma importancia ya que con ello se automatizan las pruebas generales a la aplicación web ahorrando tiempo y detectando errores antes de la puesta en producción.

Justificación Práctica

El presente proyecto se realiza por que existe la necesidad de mejorar la calidad de cada actualización en la aplicación y así mejorar la imagen de la empresa.

Justificación Social

Con este proyecto se busca apoyar al analista funcional al realizar pruebas funcionales automatizadas y así detectar de forma oportuna incidencias en el entorno de desarrollo y por consiguiente reducir el número de incidencias en producción con lo cual se tendrá usuarios satisfechos con el servicio.

1.5.2 Importancia

La importancia del presente proyecto de tesis radica en automatizar las pruebas funcionales de la aplicación web con el fin de realizar pruebas integrales a la aplicación cada vez que se realice una modificación por un desarrollo de un nuevo requerimiento o cuando se realice el desarrollo de una corrección de un fallo del sistema dado de que por falta de tiempo para realizar dichas pruebas muchas veces se realizan las pruebas solo del requerimiento y/o corrección del fallo o error mas no de toda la aplicación.

Muchas de las veces al realizar los cambios a nivel de código de la aplicación, así se haya modificado solo una parte de la aplicación esta puede estar vinculada a otro proceso de otro módulo de la aplicación y como este no fue validado, muchas de las veces este otro modulo incurre en un fallo, lo que genera la insatisfacción por parte del cliente.

Al aplicar la herramienta Badboy, cada vez que se realice modificaciones al sistema y al tener todas las pruebas funcionales automatizadas, el analista funcional y/o Release manager realiza las validaciones de la aplicación web verificando el correcto funcionamiento de la misma.

Las ventajas:

- ✓ Reducir el tiempo en ejecutar las pruebas
- ✓ Mejorar la calidad de cada entregable

Por lo tanto, implementar Badboy para la automatización de pruebas funcionales en el área de Software Factory beneficia a:

- ✓ la empresa por que mejora su imagen frente a sus clientes.

- ✓ El cliente, porque cada vez que se realice una actualización del sistema por una mejora en la aplicación o un nuevo requerimiento no afecta la continuidad del negocio debido a una falla en la aplicación.

1.6 Limitaciones de la Investigación

Para el desarrollo del presente proyecto de tesis se tiene las siguientes limitaciones:

- ✓ Tiempo para la investigación, ya que el investigador no cuenta con el tiempo necesario durante la jornada laboral debido a las actividades diarias y solo cuenta con disponibilidad después de la jornada laboral.

- ✓ Tiempo de implementación de la herramienta Badboy, ya que la aplicación web que se va a automatizar las pruebas funcionales cuenta con múltiples funcionalidades, por lo cual se está considerando la implementación en una sola funcionalidad.

CAPÍTULO II: MARCO TEÓRICO

2.1 Antecedentes de la Investigación

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

Autor: Polanco Paredes Karen Dayan

Centro de Estudio: Universidad católica de Santa María

Ciudad /País: Perú

http://tesis.ucsm.edu.pe/repositorio/bitstream/handle/UCSM/4649/71.0523.IS.pdf?sequence=1&isAllowed=y

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

La calidad del software puede medirse después de elaborado el producto. Pero esto puede resultar muy costoso si se detectan

problemas deriva dos de imperfecciones en el diseño, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida del software.

¿Cómo obtener un software de calidad?

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

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

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

Cualquier producto de ingeniería se puede probar de dos formas:

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

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

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

- ✓ Funciones incorrectas o ausentes
- ✓ Errores en la interface
- ✓ Errores en estructuras de datos o en accesos a bases de datos externas
- ✓ Errores de rendimiento
- ✓ Errores de inicialización y de terminación (Polanco Paredes, 2014)

Título: Estrategia Operativa para pruebas de Automatización y Rendimiento
Autor: Gilberto Martínez Taleno
Centro de Estudio: Tecnológico de Costa Rica
Ciudad /País: Costa Rica
https://repositoriotec.tec.ac.cr/bitstream/handle/2238/4019/Estrategia%20operativa%20para%20pruebas%20de%20automatizaci%C3%B3n%20y%20rendimiento.pdf?sequence=1&isAllowed=y

Las pruebas de software son organizadas mediante casos de pruebas, los cuales son un conjunto de condiciones o variables bajo las cuales se determinará si el requerimiento a probar de la aplicación es satisfactorio o no. Son utilizadas con el propósito de mejorar la calidad mediante la realización de tareas de verificación y validación, permitiendo conocer la calidad del producto y simplificando la detección y corrección de posibles fallos.

Dichos beneficios se pueden aprovechar para actuar a tiempo y así reducir costos de mantenimiento y de soporte hacia usuarios insatisfechos, además de la obtención de información útil que ayudará a futuros proyectos.

Existen diversas estrategias de pruebas de software, entre las cuales se encuentran:

a) Pruebas unitarias.

b) Pruebas de integración.

c) Pruebas de sistema.

d) Pruebas de aceptación y validación.

Selenium Web Driver es una tecnología libre desarrollada con el objetivo de automatizar pruebas brindando una serie de funcionalidades que permiten el manejo total del navegador, tanto en ambientes web como móvil. Por su parte JUnit pone a disposición un entorno con una gran cantidad de métodos que permiten desarrollar, ejecutar y obtener resultados de casos de pruebas automáticos. (Martinez Taleno, 2012)

Título: Desarrollo de un Modelo de Pruebas Funcionales de software basado en la Herramienta Selenium

Autor: Chinarro Morales Evelyn, Ruiz Rivera María Elena, Ruiz Lizama Edgar

Centro de Estudio: Universidad Nacional Mayor de San Marcos

Ciudad /País: Lima – Perú

http://revistasinvestigacion.unmsm.edu.pe/index.php/idata/article/view/13498/11952

Conforme se ha ido avanzando en el tiempo y con el avance de la tecnología se inició la utilización masiva de test para garantizar el cumplimiento con la especificación de los requerimientos dados por el usuario, dichos test se realizan al final del desarrollo del software. Estos test se convierten en Casos de Prueba que se aplican a los productos desarrollados para encontrar errores y corregirlos.

En la actualidad, se propone y describe una metodología que integra análisis, revisión y testing en el ciclo de vida del desarrollo de software. Las pruebas de Software empiezan a integrarse en las diferentes metodologías del desarrollo.

El objetivo general de las pruebas de software es confirmar o asegurar la calidad de los sistemas.

Las pruebas de software son un elemento crítico para la garantía de la calidad del software y representan una revisión final de las especificaciones, del diseño y de la codificación.

Para implantar con éxito una estrategia de prueba de software es necesario abordar los puntos siguientes: especificar los requisitos del producto de manera cuantificable mucho antes que comiencen las pruebas, establecer los objetivos de la prueba de manera explícita, comprender que usuarios van a manejar el software y desarrollar un perfil para cada categoría de usuario, desarrollar un plan de pruebas, llevar a cabo revisiones técnicas formales para evaluar la estrategia de prueba y los propios casos de prueba (Gelperin, &Hetzel, 1988).

Se denominan pruebas funcionales o Functional Testing, a aquellas que tienen por finalidad: 1. Identificar inconsistencias, 2. Asegurar requisitos funcionales, 3. Reducir costos de no conformidades, 4. Evitar reprocesos, 5. Mejorar la productividad, y 6. Aumentar la satisfacción del cliente, (Quality, 2016).

Las pruebas de software que tienen por objetivo probar que los sistemas desarrollados, cumplan con las funciones específicas para los cuales han sido creados, es común que este tipo de pruebas sean desarrolladas por analistas de pruebas con apoyo de algunos usuarios finales, esta etapa suele ser la última etapa de pruebas y al dar conformidad sobre esta, el paso siguiente es el pase a producción.

A este tipo de pruebas se les denomina también pruebas de comportamiento o pruebas de caja negra, ya que los testers o analistas de pruebas, no enfocan su atención a como se generan las respuestas del sistema, básicamente el enfoque de este tipo de prueba se basa en

> el análisis de los datos de entrada y en los de salida, esto generalmente se define en los casos de prueba preparados antes del inicio de las pruebas.
>
> Las pruebas son los procesos de ejecución de un programa en los que se intenta descubrir errores. Esto supone un cambio de punto de vista; nos cambia la idea que tenemos de que una prueba es exitosa si no consigue errores. Para poder diseñar casos de prueba que proporcionen un buen resultado, el ingeniero de software debe tener en cuenta los principios de las pruebas. (Chinarro Morales, Ruiz Rivera, & Ruiz Lizama, 2017)

Título: Testing y Calidad de Software. Automatización de pruebas con Selenium WebDriver
Autor: Rafael Cuba Montenegro
Centro de Estudio: Universidad Politécnica de Madrid
Ciudad /País: España
http://oa.upm.es/49320/1/PFC_RAFAEL_CUBAS_MONTENEGRO.pdf

> Una de las partes importantes que componen las tareas propias de un sistema de aseguramiento de la calidad son las pruebas software.
>
> Las pruebas software son de vital importancia dentro del ciclo de vida del desarrollo de un producto. Proporcionan información valiosa sobre la calidad alcanzada y son en sí mismas un mecanismo efectivo para descubrir fallos en el funcionamiento del software.
>
> Por otro lado, es necesario entender que el proceso de pruebas por sí sólo, no aporta calidad al producto software, simplemente la confirma o no “La calidad, si no está ahí antes de comenzar la prueba, no estará cuando termine”, por lo tanto, un buen proceso de pruebas tiene por objetivo ayudar a tomar las decisiones necesarias para alcanzar el nivel de calidad exigido y aportar confiabilidad al software desarrollado.

Actualmente la automatización de pruebas es un paso fundamental para mejorar la calidad del desarrollo. Cada vez las aplicaciones se vuelven más complejas, los requisitos van variando, se van añadiendo o modificando funcionalidades, etc., con lo que aparecen gran cantidad de versiones y la confiabilidad que produce la calidad final del software desarrollado disminuye. Es en estos casos donde la automatización de pruebas cobra importancia y representa una ventaja clara a la hora de abordar el proceso de ejecución de los test.

Actualmente existen varias herramientas que permiten generar y ejecutar scripts, pequeños programas que permiten interactuar con una aplicación dada, simulando las acciones que podría realizar el usuario real. Estas herramientas ayudan a aumentar la productividad en cuanto al número de pruebas realizadas, al alcance de cada una de ellas (se pueden probar muchas más combinaciones) y a la reproducibilidad de los test (nos aseguramos de que las pruebas se realizan siempre del mismo modo).

Estas pruebas automáticas no sustituyen al tester tradicional, persona física que realiza manualmente las pruebas, sino que ayudan a realizar tareas tediosas y repetitivas, siendo necesario un perfil más técnico, ya que exige poseer conocimientos profundos en diferentes lenguajes de programación, bases de datos, arquitectura de sistemas web, etc.

En el ámbito de las pruebas software, cuando utilizamos el termino automatizar, nos referimos a conseguir que las pruebas que se realizan de forma manual puedan ser ejecutadas de forma desatendida (sin intervención del tester), por medio de alguna herramienta que realice el proceso automáticamente.

La automatización de pruebas puede traer consigo una serie de beneficios que podrían ayudar a elevar el nivel de calidad del producto software y también a disminuir su coste.

También hay que tener en cuenta diversos factores antes de decidir si merece la pena invertir tiempo y dinero en automatizar pruebas. Si el proceso de automatización no se realiza adecuadamente muchos de los posibles beneficios que en teoría debería aportar se pueden volver en nuestra contra. (Cuba Montenegro)

Título: Automatización de Pruebas de software web basada en las reglas de negocio
Autor: Juan Camilo Salazar Rodríguez
Centro de Estudio: Universidad Distrital Francisco José de caldas
Ciudad /País: Bogotá -Colombia
http://repository.udistrital.edu.co/bitstream/11349/5194/1/SalazarRodriguezJuanCamilo2016.pdf

La fase de pruebas del ciclo de vida del Software es definida como "un proceso o una serie de procesos diseñados para asegurar que el código cumple para lo que está diseñado y no haga nada de forma involuntario" Myers y Sandler, 2004. Para lograr realizar la verificación del código contra su diseño (y por tanto, contra los requisitos del cliente), se toman datos de ejemplo de entrada, y se plantean salidas esperadas de acuerdo a los requerimientos. Para realizar una fase de pruebas correctamente, se recomienda responder las siguientes preguntas, antes inclusive de contratar al equipo de pruebas Black, 2009:

- ✓ ¿Que podría probar?
- ✓ ¿Qué debería probar?
- ✓ ¿Qué puedo probar?

Automatización de Pruebas

La automatización de pruebas puede ser definida como" el uso de software para controlar la ejecución de pruebas, la comparación de las salidas actuales con salidas predecidas, la configuración de las precondiciones, y otras funciones de control y reportes" Amaricai y

Constantinescu, 2014, pg. 1. De ahí que las pruebas automatizadas generan como resultado scripts que a su vez generan reportes sobre la efectividad de las pruebas ejecutadas sobre el Software Bajo Prueba (Software Under Test SUT).

El uso de la automatización de pruebas ayuda en gran medida al personal de Calidad. Lo anterior dado que por lo general las organizaciones asumen que cuando un software pasa a la fase de pruebas, en dicha fase serán detectados todos los bugs posibles, lo cual en la práctica es imposible, dado la ausencia de los recursos necesarios para hacerlo. Por tanto, con el uso de tests automáticos los testers podrán dedicar más tiempo a escribir más casos de prueba para cubrir más aspectos del sistema, y delegar todo el tema de validación y evaluación a los scripts de prueba Yu y Patil, 2007. De igual manera, el uso de scripts de prueba mejora la calidad de las pruebas, originado por la mínima intervención humana que requieren durante su ejecución. Gojare, Joshi y Gaigaware, 2015.

Framework de Automatización

Un framework de automatización es definido como un "conjunto de conceptos, procesos, procedimientos y entornos abstractos en los cuales los tests automáticos son diseñados, creados e implementados" Amaricai y Constantinescu, 2014, pág. 2. Además, un framework proporciona "guías, estándares de código, (...), prácticas, jerarquías de código, modularidad y mecanismos de reporte" Joy y Singh, 2015, pág. 1 a los ingenieros y personas encargadas de la implementación de la automatización de pruebas. La funcionalidad central de estos frameworks va enfocada a generar logs y reportes de la ejecución de los scripts de automatización, además de permitir extender su funcionalidad agregando librerías.

El uso de un framework de automatización de pruebas posee las siguientes ventajas Joy y Singh, 2015, pág. 1:

- ✓ Reusabilidad del código
- ✓ Costo mínimo de mantenimiento
- ✓ Intervención manual mínima
- ✓ Facilidad en la generación de reportes (Salazar Rodriguez, 2016)

2.2 Bases Teóricas referentes al Objetivo de la Investigación

Proceso de trabajo entre el Desarrollo y las Pruebas Funcionales de Software

Ciclo de vida de Software

A continuación, Pressman define el ciclo de vida clásico (modelo de cascada):

Modelo De Cascada

Hay veces en las que los requerimientos para cierto problema se comprenden bien: cuando el trabajo desde la comunicación hasta el despliegue fluye en forma razonablemente lineal.

Esta situación se encuentra en ocasiones cuando deben hacerse adaptaciones o mejoras bien definidas a un sistema ya existente (por ejemplo, una adaptación para software de contabilidad que es obligatorio hacer debido a cambios en las regulaciones gubernamentales).

También ocurre en cierto número limitado de nuevos esfuerzos de desarrollo, pero sólo cuando los requerimientos están bien definidos y tienen una estabilidad razonable.

El modelo de la cascada, a veces llamado ciclo de vida clásico, sugiere un enfoque sistemático y secuencial para el desarrollo del software, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue, para concluir con el apoyo del software terminado. (Ver Figura 04) (Pressman, págs. 33,34)

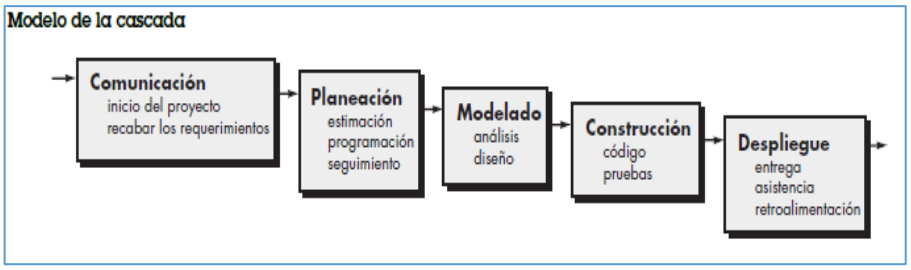

Figura 04: ciclo de vida clásico (Modelo de Cascada)
Fuente: Pressman

Sommerville, menciona y define cada una de las etapas del ciclo de vida del desarrollo de software:

> Las principales etapas de este modelo se transforman en actividades fundamentales de desarrollo:
>
> **Análisis y definición de requerimientos.** Los servicios, restricciones y metas del sistema se definen a partir de las consultas con los usuarios. Entonces, se definen en detalle y sirven como una especificación del sistema.
>
> **Diseño del sistema y del software.** El proceso de diseño del sistema divide los requerimientos en sistemas hardware o software. Establece una arquitectura completa del sistema. El diseño del software identifica y describe las abstracciones fundamentales del sistema software y sus relaciones.
>
> **Implementación y prueba de unidades.** Durante esta etapa, el diseño del software se lleva a cabo como un conjunto o unidades de programas. La prueba de unidades implica verificar que cada una cumpla su especificación.
>
> **Integración y prueba del sistema.** Los programas o las unidades individuales de programas se integran y prueban como un sistema completo para asegurar que se cumplan los requerimientos del software. Después de las pruebas, el sistema software se entrega al cliente.

Funcionamiento y mantenimiento. Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida. El sistema se instala y se pone en funcionamiento práctico. El mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema y resaltar los servicios del sistema una vez que se descubren nuevos requerimientos. (Ver Figura 05) (Sommerville, pág. 62)

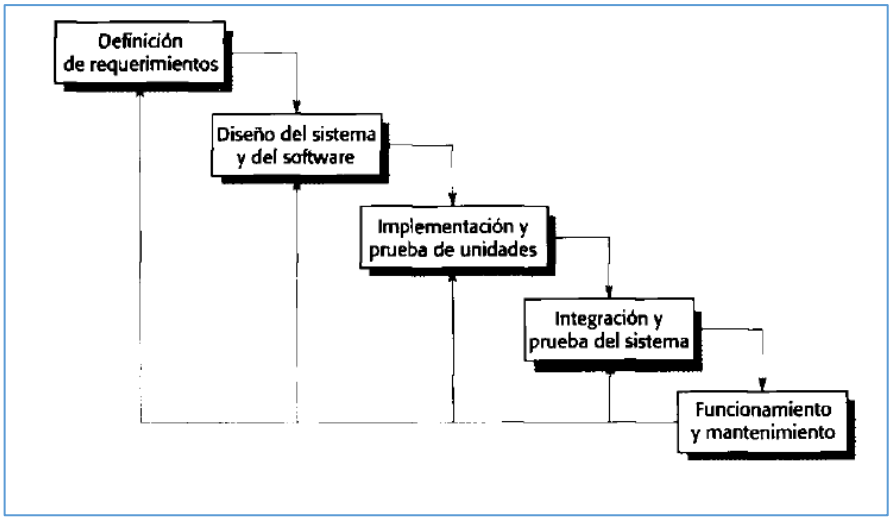

Figura 05: Ciclo de vida de Software
Fuente: Sommerville

Pruebas de Software

Se define prueba como:

"Una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas (configuración de la prueba), registrándose los resultados obtenidos." (Wong, 2017)

Sommerville, menciona que:

"Las pruebas solo pueden demostrar la presencia de errores en un programa, no pueden demostrar que no hay defectos." (Sommerville, pág. 515)

Pruebas en Aplicaciones Web

Pressman en su libro Ingeniería de Software hace un análisis detallado de las pruebas en aplicaciones web.

La prueba de una web App es una colección de actividades relacionadas con una sola meta:

- ✓ Descubrir errores en el contenido
- ✓ Función
- ✓ Utilidad
- ✓ Navegabilidad
- ✓ Rendimiento,
- ✓ Capacidad
- ✓ Seguridad de esa aplicación

Para lograr esto, se aplica una estrategia de prueba que abarca tanto revisiones como pruebas ejecutables.

¿Por qué es importante?

Si los usuarios finales encuentran errores que derrumben su fe en la web App, irán a algún otro lado en busca del contenido y de la función que necesitan, y la aplicación fracasará. Por esta razón, debe trabajarse para eliminar tantos errores como sea posible antes de poner en línea la web App.

¿Cuáles son los pasos?

El proceso de prueba de una web App comienza enfocándose en los aspectos visibles para el usuario de la aplicación y avanza hacia pruebas que ejercitan la tecnología y la infraestructura. Se realizan siete pasos durante la prueba:

- ✓ Prueba de contenido
- ✓ Prueba de interfaz

- ✓ Prueba de navegación
- ✓ Prueba de componente,
- ✓ Prueba de configuración
- ✓ Prueba de rendimiento
- ✓ Prueba de seguridad

Errores dentro de un entorno de web App

Los errores que se encuentran como consecuencia de una prueba exitosa de una web App tienen algunas características únicas:

- ✓ Puesto que muchos tipos de pruebas de webapps descubren problemas que se evidencian primero en el lado del cliente (es decir, mediante una interfaz implantada en un navegador específico o en un dispositivo de comunicación personal), con frecuencia se ve un síntoma del error, no el error en sí.
- ✓ Puesto que una webapp se implanta en algunas configuraciones distintas y dentro de diferentes entornos, puede ser difícil o imposible reproducir un error afuera del entorno en el que originalmente se encontró.
- ✓ Aunque algunos errores son resultado de diseño incorrecto o codificación HTML (u otro lenguaje de programación) impropia, muchos errores pueden rastrearse en la configuración de la webapp.
- ✓ Dado que las webapps residen dentro de una arquitectura cliente-servidor, los errores pueden ser difíciles de rastrear a través de tres capas arquitectónicas: el cliente, el servidor o la red en sí.
- ✓ Algunos errores se deben al entorno operativo estático (es decir, a la configuración específica donde se realiza la prueba), mientras que otros son atribuibles al entorno operativo dinámico (es decir, a la carga de recurso instantánea o a errores relacionados con el tiempo).

Panorama del proceso de Prueba

El proceso de prueba de webapps comienza con pruebas que ejercitan la funcionalidad del contenido y la interfaz que son inmediatamente visibles para el usuario final. Conforme avanza la prueba, se ejercitan aspectos de la arquitectura del diseño y de la navegación.

Finalmente, la atención se centra en las pruebas que examinan las capacidades tecnológicas que no siempre son aparentes para los usuarios finales: los temas de infraestructura e instalación/ implantación de la webapp.

En la figura 06 yuxtapone el proceso de prueba de la webapp con la pirámide de diseño para este tipo de aplicaciones. Observe que, conforme el flujo de la prueba avanza de izquierda a derecha y de arriba abajo, los elementos visibles para el usuario del diseño de la webapp (elementos superiores de la pirámide) se prueban primero, seguidos por los elementos de diseño de infraestructura.

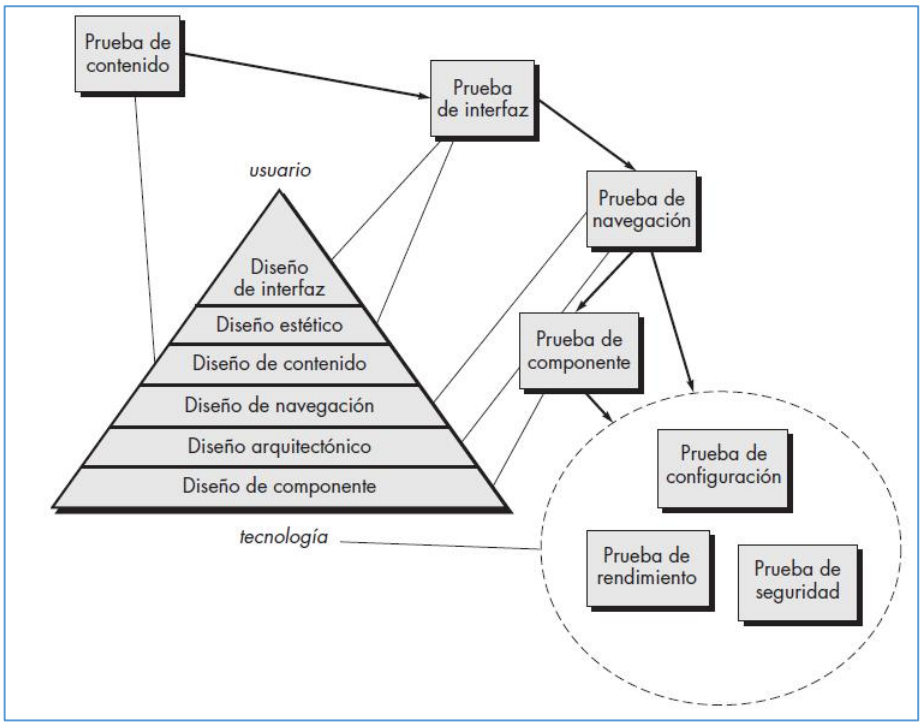

Figura 06: Proceso de Prueba
Fuente: Pressman

El proceso de prueba abarca siete tipos diferentes de pruebas.

- ✓ La prueba de contenido (y las revisiones) se enfocan en varias categorías de contenido. La intención es descubrir errores semánticos y sintácticos que afectan la precisión del contenido o la forma en la que se presenta al usuario final.
- ✓ La prueba de interfaz ejercita los mecanismos de interacción que permiten al usuario comunicarse con la webapp y valida los aspectos estéticos de la interfaz. La intención es descubrir errores que resultan a partir de mecanismos de interacción pobremente implantados o de omisiones, inconsistencias o ambigüedades en la semántica de la interfaz.
- ✓ Las pruebas de navegación aplican casos de uso, derivados de la actividad de modelado, en el diseño de casos de prueba que ejercitan cada escenario de uso contra el diseño de navegación. Los mecanismos de navegación se ponen a prueba para garantizar que cualquier error que impida completar un caso de uso se identifique y corrija.
- ✓ La prueba de componente ejercita las unidades de contenido y funcionales dentro de la webapp.
- ✓ La prueba de configuración intenta descubrir errores y/o problemas de compatibilidad que son específicos de un entorno cliente o servidor particular. Entonces se realizan pruebas para descubrir errores asociados con cada posible configuración.
- ✓ Las pruebas de seguridad incorporan una serie de pruebas diseñadas para explotar las vulnerabilidades en la webapp y su entorno. La intención es encontrar huecos de seguridad.
- ✓ La prueba de rendimiento abarca una serie de pruebas que se diseñan para valorar el tiempo de respuesta y la confiabilidad de la webapp conforme aumenta la demanda en la capacidad de recursos en el lado servidor. (Pressman, págs. 453-455-475)

Estrategia de Pruebas de software

Una estrategia de prueba de software proporciona una guía que describe los pasos que deben realizarse como parte de la prueba, cuándo se planean y se llevan a cabo dichos pasos, y cuánto esfuerzo, tiempo y recursos se requerirán.

Por tanto, cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de la prueba y la recolección y evaluación de los resultados.

¿Cuáles son los pasos?

La prueba comienza “por lo pequeño” y avanza “hacia lo grande”. Es decir que las primeras etapas de prueba se enfocan sobre un solo componente o un pequeño grupo de componentes relacionados y se aplican pruebas para descubrir errores en los datos y en la lógica de procesamiento que se encapsularon en los componentes.

Después de probar éstos, deben integrarse hasta que se construya el sistema completo. En este punto, se ejecuta una serie de pruebas de orden superior para descubrir errores en la satisfacción de los requerimientos del cliente. Conforme se descubren, los errores deben diagnosticarse y corregirse usando un proceso que se llama depuración.

Una estrategia para la prueba de software debe incluir pruebas de bajo nivel, que son necesarias para verificar que un pequeño segmento de código fuente se implementó correctamente, así como pruebas de alto nivel, que validan las principales funciones del sistema a partir de los requerimientos del cliente.

Una estrategia debe proporcionar una guía para el profesional y un conjunto de guías para el jefe de proyecto.

> Puesto que los pasos de la estrategia de prueba ocurren cuando comienza a aumentar la presión por las fechas límite, el avance debe ser medible y los problemas deben salir a la superficie tan pronto como sea posible. (Pressman, pág. 383)

Al momento de realizar las pruebas al software se debe considerar 2 tipos de procesos que Pressman detalla en su libro:

- ✓ Verificación

  "Conjunto de tareas que garantizan que el software implementa correctamente una función específica." (Pressman, pág. 384)

- ✓ Validación

  "Es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos del cliente." (Pressman, pág. 384)

La verificación responde a la pregunta ¿Estoy desarrollando el producto correcto? , mientras que la validación responde a la pregunta ¿Se está construyendo el producto de acuerdo al requerimiento del cliente?

Método Para Realizar Pruebas de Software

Para la ejecución de pruebas de software existen 2 tipos de pruebas:

- ✓ Pruebas de Caja Blanca
- ✓ Pruebas de Caja Negra

Pruebas de Caja Blanca

Las pruebas de caja blanca se basan en el conocimiento de la estructura interna del software, es decir, como este está construida en base a código.

Pressman define la prueba de caja blanca como:

Una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que:

- ✓ garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez.
- ✓ revisen todas las decisiones lógicas en sus lados verdadero y falso
- ✓ ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas
- ✓ revisen estructuras de datos internas para garantizar su validez.

(Pressman, 2010, pág. 414)

A su vez Sommerville define este tipo de pruebas como:

Una aproximación al diseño de casos de prueba en donde las pruebas se derivan a partir del conocimiento de la estructura e implementación del software. Esta aproximación se denomina a veces pruebas de «caja blanca», de «caja de cristal» o de «caja transparente» para distinguirlas de las pruebas de caja negra. (Ver Figura 07)

La comprensión del algoritmo utilizado en un componente puede ayudar a identificar particiones adicionales y casos de prueba. (Sommerville, 2005, pág. 509)

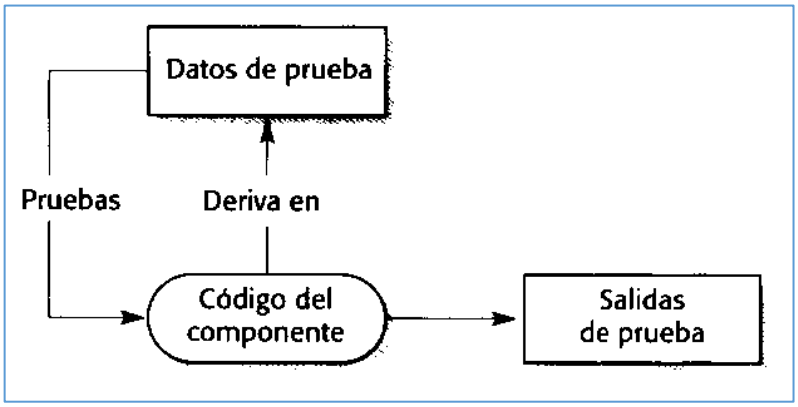

Figura 07: Pruebas estructurales
Fuente: Pressman

Pruebas de Caja Negra

Las pruebas de caja negra están orientadas a la funcionalidad del software en sí. Para realizar este tipo de pruebas se ingresan registros o datos de entrada en la aplicación devolviendo este algún resultado esperado según la especificación del requerimiento.

Definición según Pressman

> Las pruebas de caja negra, también llamadas pruebas de comportamiento, se enfocan en los requerimientos funcionales del software; es decir, las técnicas de prueba de caja negra le permiten derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa.
>
> Las pruebas de caja negra no son una alternativa para las técnicas de caja blanca. En vez de ello, es un enfoque complementario que es probable que descubra una clase de errores diferente que los métodos de caja blanca.
>
> Las pruebas de caja negra intentan encontrar errores en las categorías siguientes:
>
> - ✓ funciones incorrectas o faltantes.
> - ✓ errores de interfaz.
> - ✓ errores en las estructuras de datos o en el acceso a bases de datos externas,
> - ✓ errores de comportamiento o rendimiento
> - ✓ errores de inicialización y terminación.
>
> (Pressman, pág. 423)

Tiempo Empleado en Pruebas Funcionales

En la figura 08 y en la figura 09 se muestra que en un entorno manual de pruebas la puesta en producción del entregable toma más tiempo mientras que en un entorno automatizado el despliegue es más rápido.

Eso no significa que no se deba realizar pruebas manuales ya que hay procesos que serían técnicamente complejos al momento de automatizar.

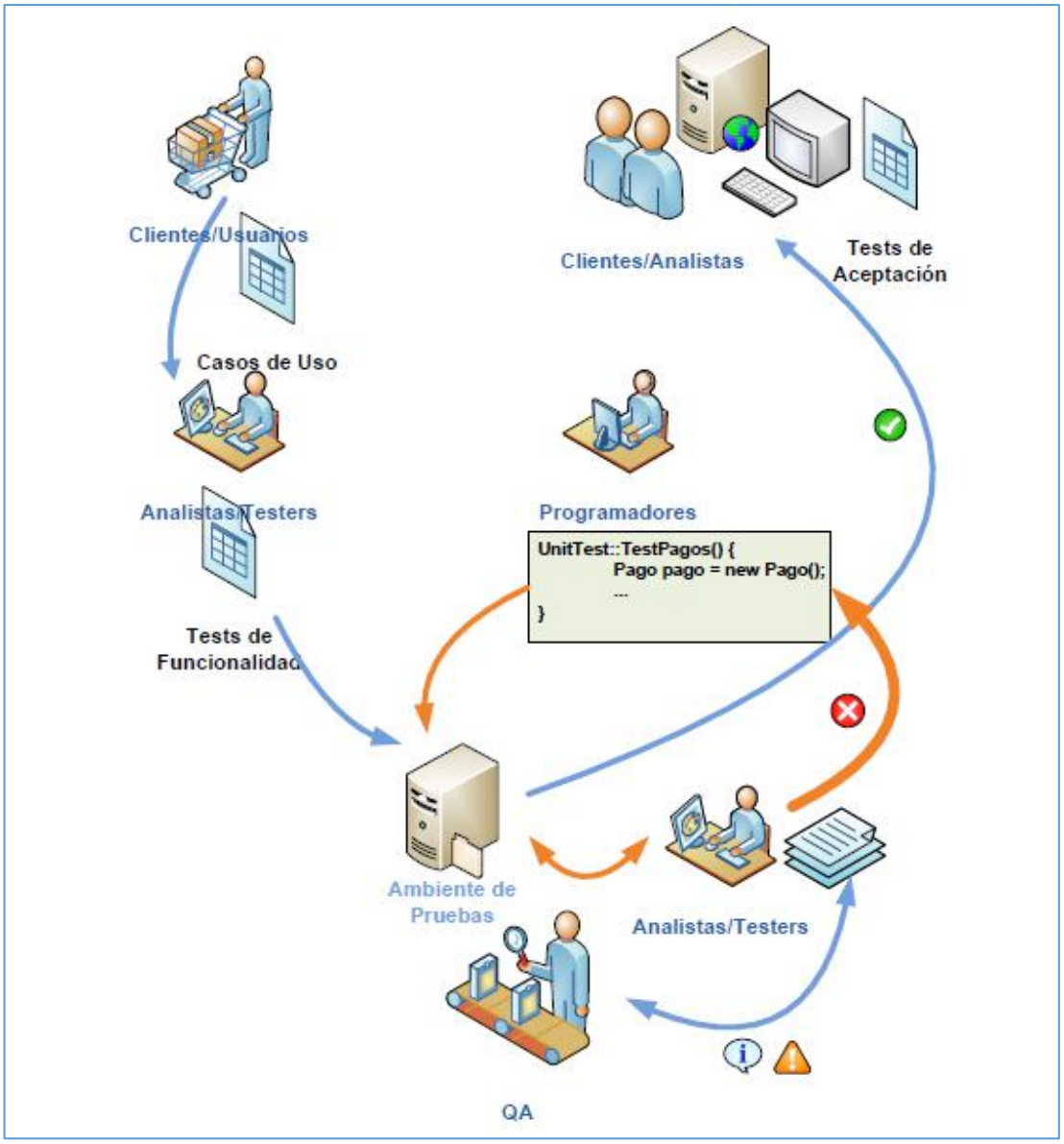

Figura 08: Trabajo con test manuales
Elaboración: IT-Mentor

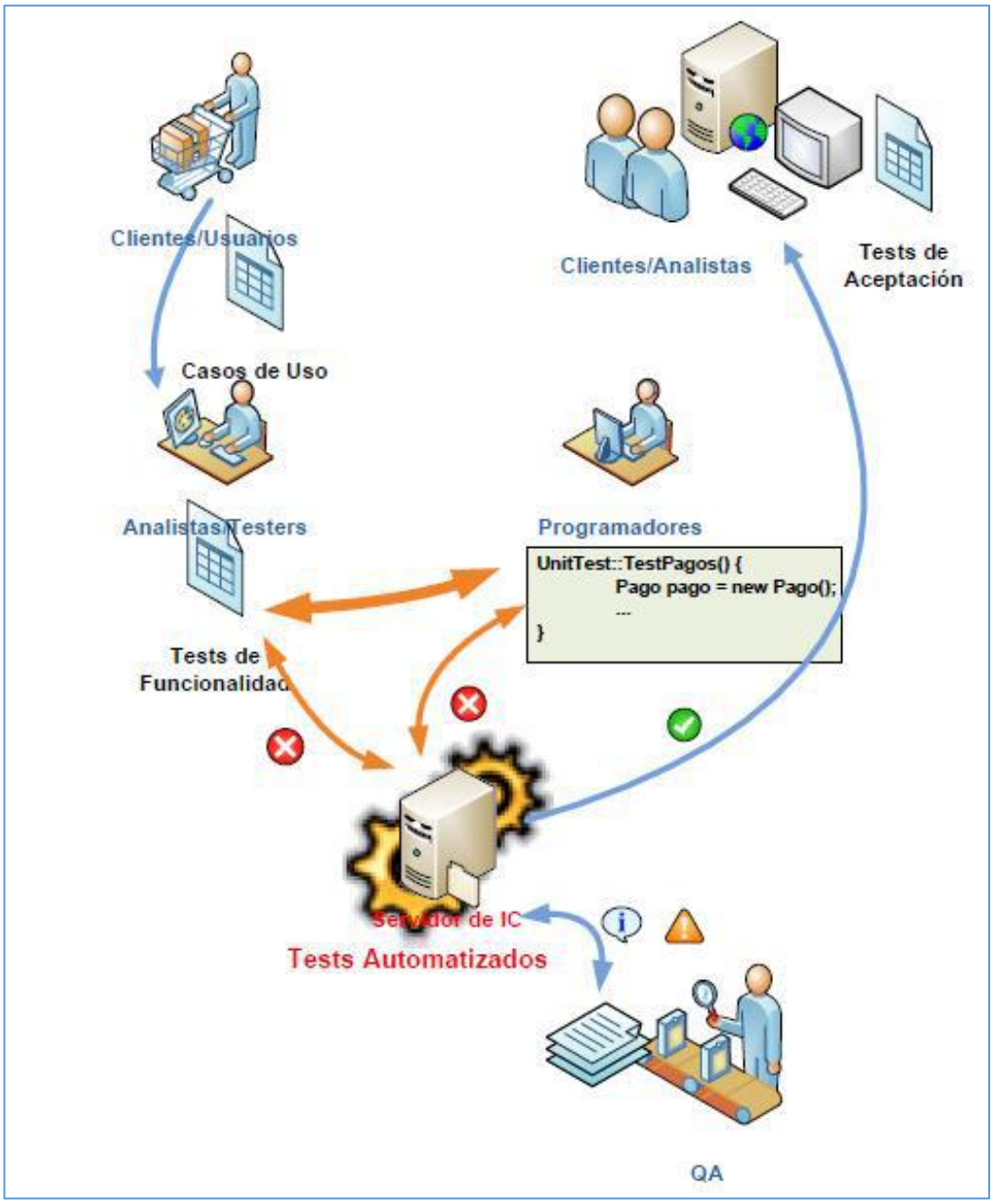

Figura 09: Trabajo con test automatizados
Elaboración: IT-Mentor

Costo en Solucionar fallas de Software

Kendall indica lo siguiente:

> Antes de poner el sistema en funcionamiento es necesario probarlo. Es mucho menos costoso encontrar los problemas antes que el sistema se entregue a los usuarios.

> Una parte de las pruebas las realizan los programadores solos, y otra la llevan a cabo de manera conjunta con los analistas de sistemas.
>
> Primero se realiza una serie de pruebas con datos de muestra para determinar con precisión cuáles son los problemas y posteriormente se realiza otra con datos reales del sistema actual.
>
> El mantenimiento del sistema de información y su documentación empiezan en esta fase y se llevan a cabo de manera rutinaria durante toda su vida útil.
>
> Gran parte del trabajo habitual del programador consiste en el mantenimiento, y las empresas invierten enormes sumas de dinero en esta actividad.
>
> Parte del mantenimiento, como las actualizaciones de programas, se pueden realizar de manera automática a través de un sitio Web.
>
> Muchos de los procedimientos sistemáticos que el analista emplea durante el ciclo de vida del desarrollo de sistemas pueden contribuir a garantizar que el mantenimiento se mantendrá al mínimo.

También indica que:

> Después de instalar un sistema, se le debe dar mantenimiento, es decir, los programas de cómputo tienen que ser modificados y actualizados cuando lo requieran.
>
> Según estimaciones, los departamentos invierten en mantenimiento de 48 a 60 por ciento del tiempo total del desarrollo de sistemas.
>
> Queda muy poco tiempo para el desarrollo de nuevos sistemas. Conforme se incrementa el número de programas escritos, también lo hace la cantidad de mantenimiento que requieren.

El mantenimiento se realiza por dos razones.

- ✓ La corrección de errores del software. No importa cuán exhaustivamente se pruebe el sistema, los errores se cuelan en los programas de cómputo. Los errores en el software comercial para PC se documentan como "anomalías conocidas", y se corrigen en el lanzamiento de nuevas versiones del software o en revisiones intermedias. En el software hecho a la medida, los errores se deben corregir en el momento que se detectan.

- ✓ El mantenimiento del sistema es la mejora de las capacidades del software en respuesta a las cambiantes necesidades de una organización, que por lo general tienen que ver con alguna de las siguientes tres situaciones:
  - Con frecuencia, después de familiarizarse con el sistema de cómputo y sus capacidades, los usuarios requieren características adicionales.
  - El negocio cambia con el tiempo.
  - El hardware y el software cambian a un ritmo acelerado. (Kendall & Kendall, págs. 13-14)

En la Figura 10 se puede deducir que en la etapa donde el costo es mayor a comparación de las otras etapas de ciclo de vida de software es cuando ya se encuentra desplegada en el entorno de producción.

Los costos para solucionar una falla en el software se dan para las siguientes condiciones:

- ✓ El tiempo que toma el programador en solucionar la falla.
- ✓ El costo de la empresa cliente al tener inoperativo el sistema.
- ✓ Mala imagen de la empresa proveedora del software.
- ✓ Acciones legales que la empresa cliente toma a la empresa proveedora del software.

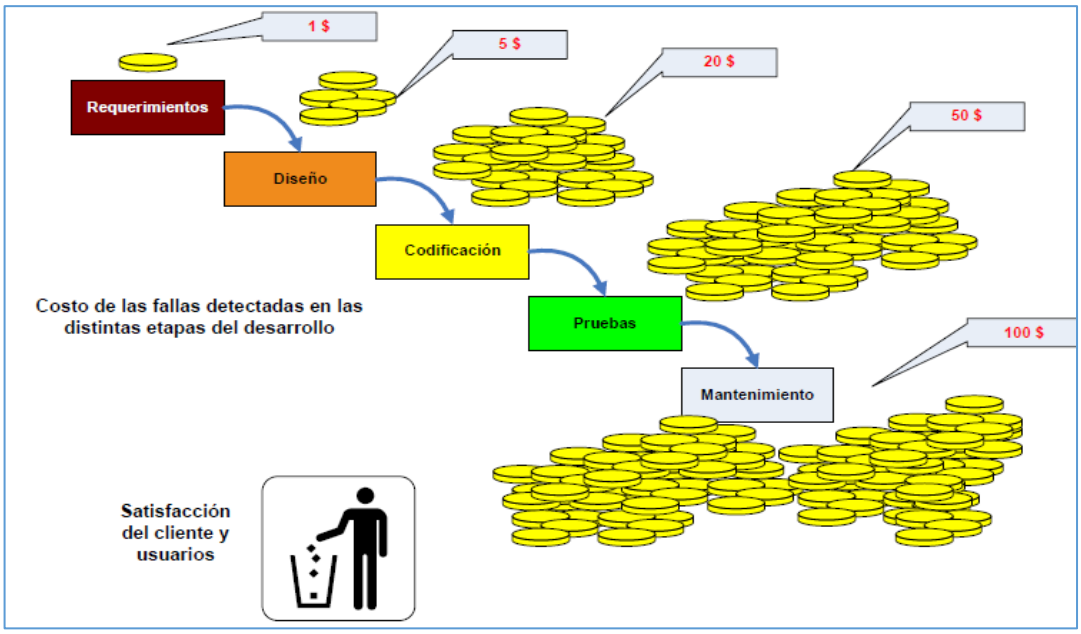

Figura 10: Costo de fallas de software en distintas etapas
Elaboración: IT-Mentor

Automatización De Pruebas

Sommerville sobre la automatización de pruebas

> Las pruebas son una fase cara y laboriosa del proceso de software. Como consecuencia, las herramientas de prueba estaban entre las herramientas de software a desarrollar.
>
> Actualmente, estas herramientas ofrecen una serie de facilidades y su uso puede reducir significativamente los costes de las pruebas.
>
> Un banco de pruebas de software es un conjunto integrado de herramientas para simular otras partes del sistema y generar datos de prueba de dicho sistema.
>
> La Figura 11 muestra algunas de las herramientas que podrían incluirse en un banco de trabajo de pruebas de este tipo:
>
> - ✓ **Gestor de pruebas.** Gestiona la ejecución de las pruebas del programa. El gestor de pruebas mantiene un registro de los

datos de las pruebas, resultados esperados y facilidades del programa que han sido probadas. Los marcos de trabajo automatizados tales como JUnit son ejemplos de gestores de pruebas.

- ✓ **Generador de datos de prueba.** Genera datos de prueba para el programa a probar. Esto puede conseguirse seleccionando datos de una base de datos o utilizando patrones para generar datos aleatorios de forma correcta.

- ✓ **Oráculo**. Genera predicciones de resultados esperados de pruebas. Los oráculos pueden ser versiones previas del programa o sistemas de prototipos. Las pruebas back-toback implican ejecutar el oráculo y el programa a probar en paralelo. Las diferencias entre sus salidas son resaltadas.

- ✓ **Comparador de ficheros.** Compara los resultados de las pruebas del programa con los resultados de pruebas previos e informa de las diferencias entre ellos. Los comparadores se utilizan en pruebas de regresión en las que se comparan los resultados de ejecutar diferentes versiones. Cuando se utilizan pruebas automatizadas, los comparadores pueden ser llamados desde las mismas pruebas.

- ✓ **Generador de informes.** Proporciona la definición de informes y facilidades de generación para los resultados de las pruebas.

- ✓ **Analizador dinámico.** Añade código a un programa para contar el número de veces que se ha ejecutado cada sentencia. Después de las pruebas, se genera un perfil de ejecución que muestra cuántas veces se ha ejecutado cada sentencia del programa.

- ✓ **Simulador**. Se pueden utilizar diferentes tipos de simuladores. Los simuladores de la máquina objetivo simulan la máquina sobre la que se ejecuta el programa. Los simuladores de interfaces de usuario son programas conducidos por scripts que simulan múltiples interacciones de usuarios simultáneas. Utilizar simuladores para Entrada/Salida implica que el comportamiento temporal de la secuencia de las transacciones es repetible. (Sommerville, págs. 513,514)

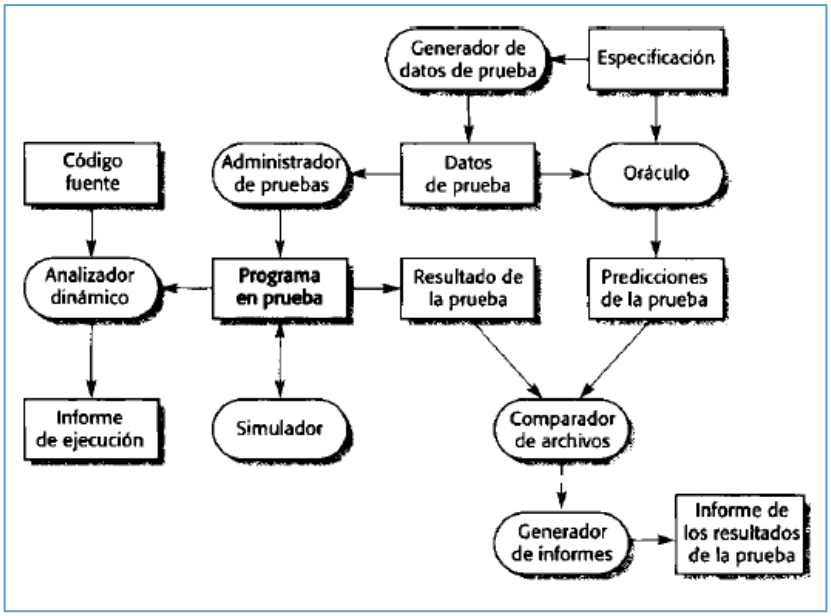

Figura 11: Banco de trabajo de pruebas
Fuente: Sommerville

Razones para Automatizar Pruebas

- ✓ Ciclo de prueba manual es muy largo
- ✓ Proceso de prueba manual es propenso a errores
- ✓ Liberar a la gente para realizar tareas creativas
- ✓ Generar un ambiente de confianza soportado por los test
- ✓ Obtener realimentación de forma temprana y con alta frecuencia
- ✓ Generar conocimiento del sistema en desarrollo a partir de los test
- ✓ Generar documentación del código consistente
- ✓ Generar una mejor utilización de los recursos a partir de menores costos (IT - mentor, pág. 9)

¿Qué debería automatizarse?

- ✓ Pruebas unitarias y de componentes
- ✓ Pruebas de funcionalidad sin interfaces de usuario
- ✓ Pruebas de sistema con interfaces de usuario (IT - mentor, pág. 9)

Framework o Herramientas de Automatización de Pruebas Funcionales

> Para la automatización de las pruebas funcionales son especialmente indicadas las herramientas de ejecución de las pruebas de captura y reproducción. Estas herramientas permiten al tester capturar y grabar pruebas, para luego editarlas, modificarlas y reproducirlas en distintos entornos.
>
> Herramientas que graban la interfaz de usuario a nivel de componentes y no de bitmaps son más útiles. Durante la grabación se capturan las acciones realizadas por el tester, creando automáticamente un script en algún lenguaje de alto nivel.
>
> Luego el tester modifica el script para crear una prueba reusable y mantenible. Este script se vuelve la línea base y luego es reproducido en una nueva versión, contra la cual es comparado.
>
> En general estas herramientas vienen acompañadas de un comparador, que compara automáticamente la salida en el momento de ejecutar el script con la salida grabada**.** (Esmite, Farias, Farias, & Perez)

Stefany Rodríguez en una publicación web indica que:

> La automatización de prueba es ventajosa en situaciones en las cuales el software se modifica constantemente, dado que hasta las modificaciones menores pueden ocasionar que funcionalidad ya desarrollada deje de funcionar**.**

Para la realización de pruebas funcionales existen diversas herramientas, a continuación, Stefany Rodríguez detalla algunas:

- ✓ **Selenium**

  Es un framework para pruebas de aplicaciones Web, descargable de forma gratuita desde su sitio web. Proporciona una herramienta de grabación y playback, que permite desarrollar pruebas sin necesidad de aprender un lenguaje de Scripting. Incluye características como grabación, playback, selección de campos, auto completar formularios, pruebas de recorrido (Walkthrough), debug, puntos de control, scripts ruby y otros formatos.

- ✓ **HP Quicktest Professional (QTP)**

  Proporciona la capacidad de automatizar pruebas funcionales y pruebas de regresión para software y ambientes de prueba. Proporciona la capacidad de definir Scripts de prueba y posee una interfaz gráfica que le permiten al usuario emular la funcionalidad que desea probar, incluyendo el uso de interfaces de usuario de las aplicaciones a probar. Incluye características como: Vista de experto, pruebas de procesos de negocio, grabado de pantalla (para captura de las evidencias de prueba), entre otras posibilidades.

- ✓ **Watir**

  Pronunciado "Water", es una familia de librerías Ruby de Código Abierto (Open Source) para la automatización de navegadores web. Le permite a su usuario escribir pruebas fáciles de leer y mantener. Sencilla y flexible. Tiene la capacidad de hacer clic en enlaces, llenar formularios de pantallas con datos y presionar botones. Watir también revisa los resultados, incluyendo verificar si los textos esperados se muestran en las páginas. Tiene la capacidad de enlazarse con bases de datos, leer archivos de datos y hojas de cálculo, exportar XML y estructurar los códigos como librerías reutilizables.

- ✓ **Visual Studio Test Proffessional**

  Conjunto de herramientas de pruebas integradas desarrolladas por Microsoft, que proporcionan soporte a todo el ciclo de planificación, ejecución y registro de pruebas, con facilidades de colaboración entre analistas de prueba (testers) y desarrolladores en la herramienta. Proporciona capacidad de realizar pruebas manuales, reutilización de pruebas manuales, integración con el "team foundation server", gestión de ciclo de vida de aplicaciones, entre otros.

- ✓ **Rational Functional Tester**

  Herramienta de automatización de pruebas funcionales y de regresión. Proporciona capacidades de pruebas de interfaz gráfica, pruebas manejadas por datos (Data Driven), pruebas funcionales y pruebas de regresión.

  Algunas de sus características son: Simplificación de creación y visualización de pruebas, pruebas de tipo storyboards, trazabilidad en todo el ciclo de vida, validación de data dinámica (por medio de un wizard), e inclusive capacidad de definir scripts (por medio de lenguajes de Scripting). (Rodriguez Johnson, 2016)

Luego mencionar otras herramientas utilizadas para la realización de pruebas funcionales, a continuación, se describirá a detalle la herramienta BadBoy, que es la herramienta que se está considerando para el presente proyecto.

- ✓ **BadBoy**

  Badboy es una herramienta de gran alcance diseñada para ayudar en la prueba y en el desarrollo de aplicaciones. Permite efectuar el testeo de la Web, con docenas de características incluyendo una interfaz simple, fácil e intuitiva, mediante los métodos de captura y repetición, siendo

una gran ayuda para la prueba de carga de gran alcance, informes detallados, gráficos, etc.

Badboy trae embebido el navegador Internet Explorer de Microsoft, monitorizando y controlando las acciones que se producen.

Esto permite:

- Realizar capturas de los parámetros del CGI, las páginas y los framesets peticionados.
- Modificar dichas capturas y volver a ejecutarlas en cualquier momento automáticamente.
- Grabar las capturas como scripts y compartirlos dentro del entorno de trabajo.
- Recoger las estadísticas del funcionamiento mientras se ejecuta una prueba.
- Realizar pruebas de regresión de áreas completas de sitios Web complejos con un solo clic.

Esto permite:

Badboy incluye una potente funcionalidad para permitir crear scripts, de una manera sencilla, con las navegaciones realizadas por sitios Web complejos sin la intervención del usuario. Una vez que estén creados los scripts, éstos se pueden compartir para conseguir un mayor aumento de la productividad.

Identificación de las diferentes áreas de trabajo

Al acceder a Badboy, nos encontramos con una interfaz gráfica, tal como se muestra en la figura 12

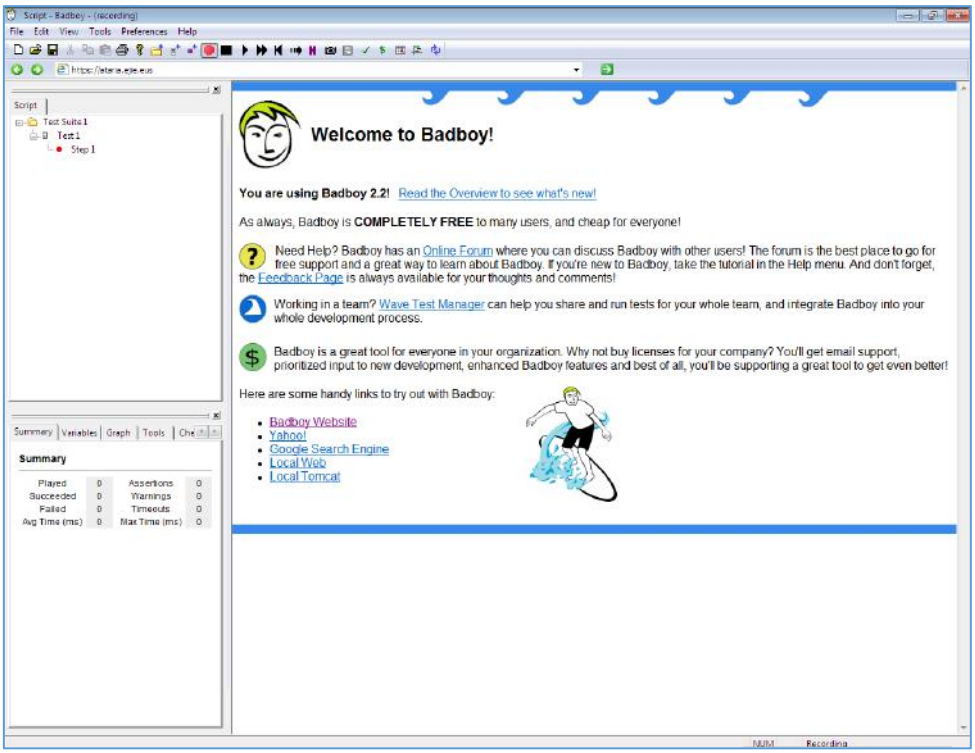

Figura 12: Interfaz gráfica badboy
Elaboración: Ander Martínez

El área de trabajo está dividida en diferentes secciones:

Sección de Menús y Accesos Rápidos: En esta sección se encuentran todas las acciones que es posible realizar por la herramienta. Ver figura 13

Figura 13: Barra de Herramientas Badboy
Elaboración: Ander Martínez

Sección de Script: Aquí se podrán ver las peticiones, las repuestas, los parámetros CGI y las distintas acciones que se hayan considerado durante la navegación. Se agrupan por: Carpeta, Prueba y Paso. Ver figura 14

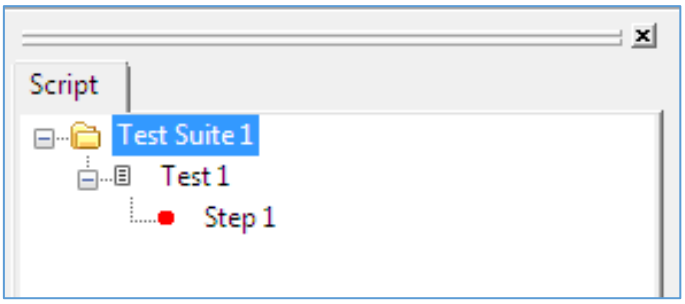

Figura 14: Sección Scripts Badboy
Elaboración: Ander Martínez

Sección de Resumen, Variables, Gráficos, Herramientas y Checks: Esta sección dispone de pestañas para poder visualizar los datos correspondientes a las diferentes acciones realizadas en el transcurso de la navegación. Ver figura 15

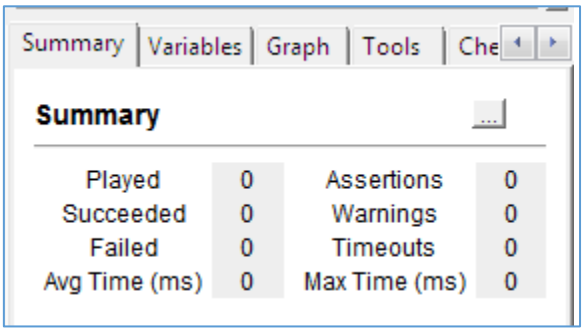

Figura 15: Sección de Resumen, variables, gráficos, herramientas y Checks Badboy
Elaboración: Ander Martínez

Sección de Navegación: Aquí se verá el contenido de la Web y es la sección desde donde se ejecutarán las diferentes acciones para la grabación de las mismas bajo la herramienta. Ver figura 16

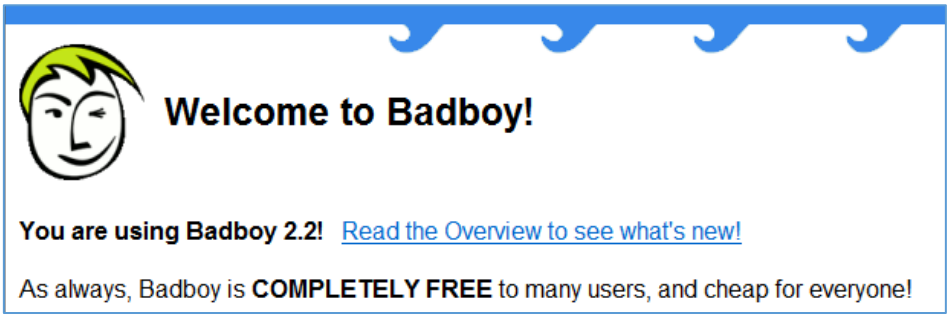

Figura 16: Sección navegación Badboy
Elaboración: Ander Martínez

Crear un nuevo proyecto de Navegación

Podremos crear un nuevo entorno de navegación pulsando el icono  existente en la sección de Menús, o bien a través del menú *File > New:*

Tal como se muestra en la figura 17

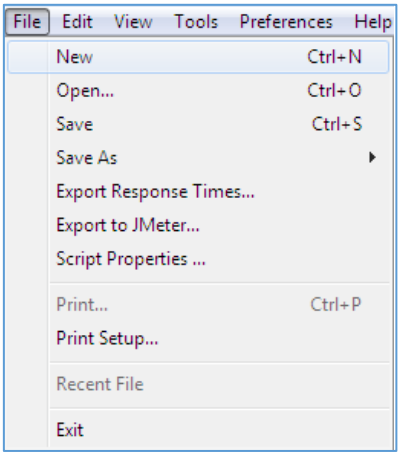

Figura 17: Menú File Badboy
Elaboración: Ander Martínez

Una vez hemos creado el nuevo proyecto, introduciremos en el campo URL. Ver figura 18

Figura 18: Sección URL Badboy
Elaboración: Ander Martínez

La dirección en donde tenemos la aplicación y sobre el cual se desea crear la grabación de la navegación.
Una vez se haya introducido, pulsaremos la tecla Intro o Enter.

Cómo grabar una navegación

Siempre por defecto disponemos del botón de grabación  activado, por lo que desde el primer momento en el que se accede al aplicativo se está grabando la navegación.

Figura 19: inicio de grabación de pruebas.
Elaboración: propia

Podemos observar que, en la sección de Script, nos aparece uno, que se corresponde con el primer acceso al aplicativo. Si seguimos navegando por el aplicativo, se irán aumentando los elementos para un paso dado. Tal como vemos en la figura 19 y figura 20.

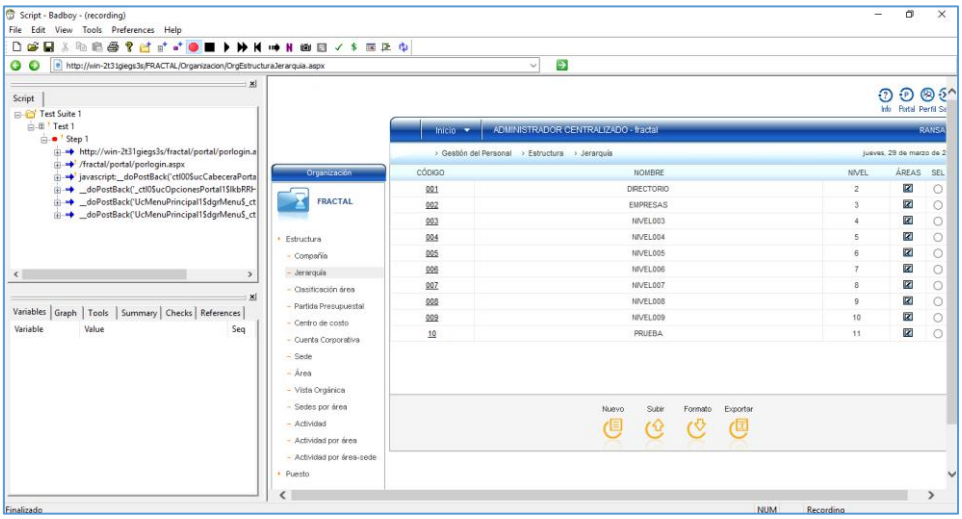

Figura 20: Grabación de secuencias de pruebas
Elaboración: propia

Una vez hemos terminado la navegación, procederemos a guardarla en disco, bien para poder ejecutarla en otra ocasión, modificarla o compartirla con otros miembros del equipo de proyecto. Para ello pulsaremos sobre el botón  .

Automatización de scripts

Ser capaz de reproducir repetidamente una secuencia de actividades de navegación puede resultar un mecanismo muy útil para depurar y probar un sitio Web.

Lamentablemente, repetir peticiones ya lanzadas anteriormente puede no satisfacer los requisitos de sitios Web complejos. Situaciones en los que esto puede ser así:

- ✓ Un identificador introducido debe ser único. Introducir algunos valores dos veces genera un error.

- ✓ Se ha grabado el script en un servidor (por ejemplo, la máquina de desarrollo local) pero se desea reproducirlo bajo uno diferente (se necesitará una manera de modificar el nombre de la máquina sobre la que lanzar las peticiones).

Badboy permite resolver estos problemas de diferentes maneras:

- ✓ Editar los parámetros.

- ✓ Utilizar variables en los *scripts*.

- ✓ Usar una gran variedad de herramientas tales como Buscar y Reemplazar.

Resultados

Tras lanzar un *script* es importante mantener un registro de lo ocurrido. Por ejemplo, es interesante conocer el tiempo medio transcurrido entre petición y respuesta, etc. Badboy captura toda esta información y hace posible acceder a ella de manera rápida a través de la vista Resumen.

La figura 21 muestra un ejemplo del registro de resultados una vez finalizada las pruebas.

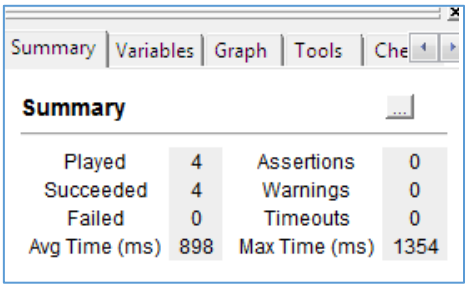

Figura 21: muestra de resultados
Elaboración: Ander Martínez

La tabla 01 describe los datos aportados en esta vista *Resumen*:

Tabla 01:
Descripción de resultados

Estadística	Descripción
Played	El número de elementos del *script* que fueron ejecutados y devolvieron una respuesta.
Succeeded	El número de elementos del *script* que fueron ejecutados y devolvieron una respuesta exitosa.
Failed	El número de elementos del *script* que fueron ejecutados y devolvieron una respuesta fallida.
Avg Time (ms)	El porcentaje de tiempo (en milisegundos) para los elementos que fueron ejecutados y devolvieron una respuesta.
Assertions	El número de aserciones fallidas.
Warnings	El número de Warnings generados.
Timeouts	El número de desconexiones que se han producido.
Max Time (ms)	El mayor tiempo transcurrido en una única respuesta.

Fuente: EJIE

Generar Informes

Ya se ha visto que la pestaña *Resumen* muestra la información aportada tras la ejecución del *script*. Sin embargo, si se desea obtener una visión más detallada de los resultados obtenidos y además sin la necesidad de tener que estar ejecutando Badboy y con la posibilidad de integrar estos resultados con programas externos o documentos, es muy interesante la posibilidad de generación de informes con los resultados obtenidos.

Informes HTML

La manera de sencilla de obtener un informe en HTML es seleccionar la opción de menú *View* > *Report*.

Lo que guardará el informe HTML generado en un archivo temporal y lo mostrará en la sección de Navegación de Badboy.

Si se desea guardar o enviar el informe generado se puede usar la opción de menú *File* > *Save as* > *HTML Report*... (Martinez, 2014)

La figura 22 muestra el aspecto de un informe HTML:

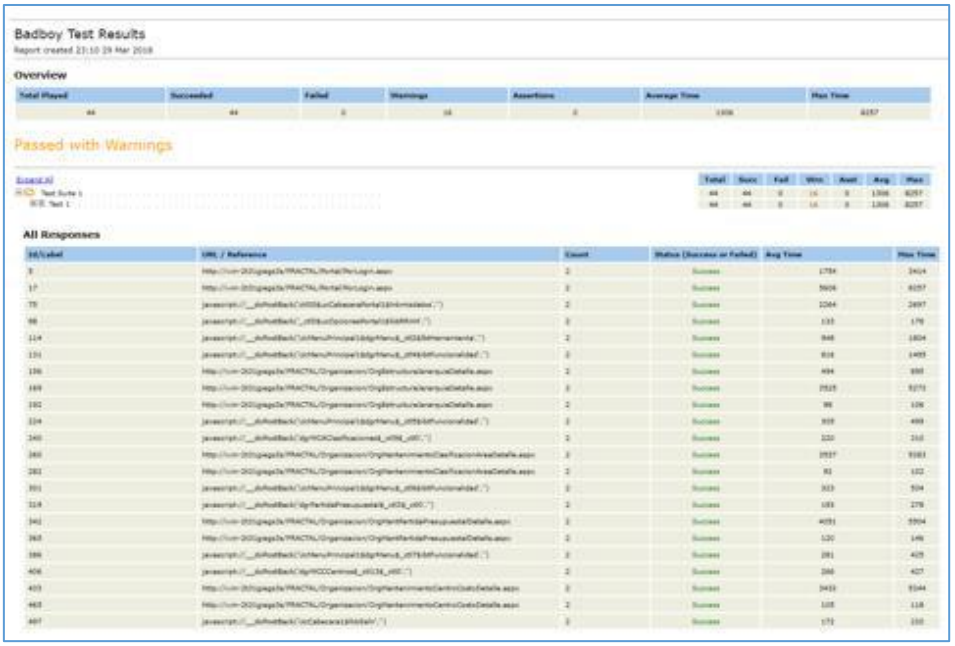

Figura 22: Resultados de pruebas
Elaboración: propia

2.3 Definición de términos básicos

Implementación

"Instalación y puesta en marcha de un sistema o conjunto de programas de utilidad para el usuario." (Gran Diccionario de la Lengua Española, 2016)

Falla

Pressman (2010) define a la falla como:

> La falta de conformidad con los requerimientos del software. Pero, incluso con esta definición, hay gradaciones. Las fallas pueden ser leves o catastróficas. Una falla podría corregirse en segundos, mientras que otra tal vez requiera de varias semanas o meses de trabajo para ser corregida. Para complicar más el asunto, la corrección de una falla quizá dé como resultado la introducción de otros errores que a su vez originen otras fallas. (pág. 376)

Costos

"Son la suma de esfuerzos y recursos que se han invertido para producir algo" (Gonzalez, 1998, pág. 2)

Desarrollo de Software

Howard Baetjer, Jr. [Bae98] se refiere al desarrollo de Software como:

> El proceso en el que se reúne el conocimiento y se incluye en el software para convertirse en software. El proceso proporciona una interacción entre los usuarios y los diseñadores, entre los usuarios y las herramientas de desarrollo, y entre los diseñadores y las herramientas de desarrollo [tecnología] citado en (Pressman, 2010, pág. 26)

Aplicación Web

"Una aplicación Web es un sistema software accesible a través de Internet o Intranet y se construyen de acuerdo con ciertas tecnologías y estándares" (Oliveros, Danyans, & Mastropietro, 2014)

Proceso

"Un proceso es un conjunto de actividades, acciones y tareas que se ejecutan cuando va a crearse algún producto del trabajo." (Pressman, 2010, pág. 12)

Reglas de negocio

> Las reglas de negocio permiten definir las condiciones para establecer un negocio o precisar de qué forma se controlará el comportamiento de los eventos dentro de este. Muy importante resulta establecer los pasos que se deben seguir para alcanzar las metas y objetivos del mismo. Las reglas de negocio son requisitos que definen cómo el negocio debe operar. (Garcia Gonzalez & Andre Ampuero, 2015, pág. 42)

Incidencia

Se denomina incidencia al error o defecto en el software que dificulta la operatividad en el sistema.

2.4 Hipótesis

2.4.1 Hipótesis General

Si se implementa la herramienta Badboy, entonces se mejorará el entorno de pruebas funcionales en aplicaciones web.

2.4.2 Hipótesis Específicas

a) Si se describe el método de pruebas funcionales del analista funcional, entonces se identificarán los fallos en el desarrollo de software.

b) Si se automatiza las pruebas funcionales de aplicaciones web, entonces se reducirá el tiempo empleado en las pruebas funcionales básicas de las reglas de negocio.

c) Si se rediseña el proceso de trabajo entre el desarrollo y las pruebas funcionales de software, entonces se reducirán los costos en solucionar fallas en el software.

2.5 Variables

2.5.1 Variables Independientes

General

- ✓ Herramienta Badboy

Específicas

- ✓ Método de pruebas funcionales
- ✓ Automatización de pruebas funcionales
- ✓ Proceso de trabajo entre el desarrollo y las pruebas funcionales

2.5.2 Variables Dependientes

General

- ✓ Pruebas funcionales en aplicaciones Web.

Específicas

- ✓ Identificación de fallos de Software
- ✓ Tiempo empleado en pruebas funcionales
- ✓ Costo en solucionar una falla en el software

2.5.3 Indicadores de las Variables Dependientes

- ✓ Cantidad de errores por tipo de actualización.
- ✓ Tiempo empleado al ejecutar pruebas funcionales.
- ✓ Costo por hora del programador/Release manager.

CAPÍTULO III: DISEÑO METODOLÓGICO

3.1 Diseño de la investigación

Marcelo Gómez nos dice en su libro Metodología de la investigación científica:

> Disponemos de distintas clases de diseños o estrategias para investigar y debemos elegir el que más se adecue al problema de investigación que nos hemos planteado. Si el diseño está bien concebido, y es coherente con las preguntas, el enfoque y el objetivo del estudio, el producto final (sus resultados) tendrán mayores posibilidades de éxito para generar conocimiento científicamente válido. Hay dos grandes tipos de diseños: los diseños experimentales y los no experimentales.
>
> **Diseños experimentales**
>
> El término "experimento", en el sentido científico del término, se refiere a una investigación en la que se manipulan intencionalmente (se obligan a cambiar de estado) una o más variables independientes (supuestas causas), para analizar las consecuencias que esa manipulación tiene sobre una o más variables dependientes (supuestos efectos), dentro de una situación de control creada por el investigador. Cuando en realidad existe una relación entre una variable independiente y una dependiente, al variar intencionalmente la primera, la segunda también variará, por ejemplo, si la motivación de los alumnos se relaciona con su rendimiento académico, al variar la motivación deberá variar el rendimiento académico (aunque no sea la única variable

interviniente) Para llevar a cabo un experimento, y que este sea válido y confiable, es necesario tener en cuenta algunos aspectos que pasamos a describir. (Gomez, 2016)

Por otra parte, Hernández, Fernández y Baptista, en su libro: Metodología de la investigación menciona que:

> Una acepción particular de experimento, más armónica con un sentido científico del término, se refiere a un estudio en el que se manipulan intencionalmente una o más variables independientes (supuestas causas antecedentes), para analizar las consecuencias que la manipulación tiene sobre una o más variables dependientes (supuestos efectos-consecuentes), dentro de una situación de control para el investigador. Esta definición quizá parezca compleja; sin embargo, conforme se analicen sus componentes se aclarara el sentido de la misma. Ver la figura 23: (Hernandez Sampieri, Fernadez Collado, & Baptista Lucio, 2014)

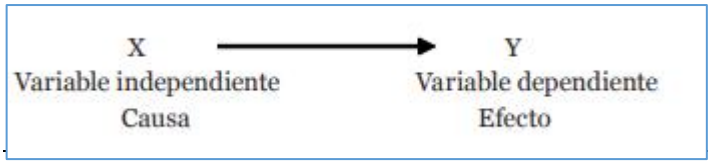

Figura 23: Variable causa-efecto
Fuente: Hernández, Fernández y Baptista

La manipulación de las variables independientes

Un experimento se lleva a cabo para analizar si una o más variables independientes afectan a una o más variables dependientes y como lo hacen. Para obtener evidencia de esta relación supuesta, el investigador manipula la variable independiente y observa si la dependiente varía o no (aquí, manipular es sinónimo de hacer variar o asignar distintos valores a la variable independiente).

> La variable independiente se manipula, la variable dependiente se mide, para ver el efecto que la manipulación de la variable independiente tiene en ella. (Gomez, 2016)

Para el presente proyecto de tesis, se está empleando el diseño experimental donde se aplica los siguientes criterios según (Campbell & Stanley, 1996)

- ✓ R = Para asignación al azar o aleatorización.
- ✓ G = Para determinar el número o grupo de sujetos (G1 = grupo 1; …)
- ✓ X = Para la presencia de un tratamiento, estímulo, o condición experimental, las cuales pueden ser varias, por lo que puede tener subíndices (X_1, X_2, …, Xn)
- ✓ O = Una medición a los sujetos de un grupo. Puede tener subíndices para determinar el número de mediciones (O_1, O_2, ..., On)
- ✓ – = Ausencia de un estímulo, indica que se trata de un grupo control.

En la tabla 02 se describe el diseño experimental para cada una de las hipótesis:

Tabla 02:
Experimentos

1°	$\mathbf{G_1}$	$\boldsymbol{O_1}$	**X**	$\boldsymbol{O_2}$
2°	$\mathbf{G_1}$	$\boldsymbol{O_1}$	**X**	$\boldsymbol{O_2}$
3°	$\mathbf{G_1}$	$\boldsymbol{O_1}$	**X**	$\boldsymbol{O_2}$

Fuente: Elaboración propia

En la tabla 02 se desprende lo siguiente:

El experimento el grupo de personas a analizar son del área software Factory $\mathbf{G_1}$ a los cuales se le aplicara una medición antes de implementar la herramienta BadBoy $\boldsymbol{O_1}$, posteriormente después de implementada la herramienta **X** se volverá a realizar la medición de resultados $\boldsymbol{O_2}$.

3.2 Tipo de investigación

A continuación, Hugo Sánchez y Carlos Reyes definen el tipo de investigación Tecnológica

> **Investigación tecnológica**
>
> Responde a problemas técnicos, está orientada a demostrar la validez de ciertas técnicas bajo las cuales se aplican principios científicos que demuestren su eficacia en la modificación o trasformación de un hecho o fenómeno. La investigación tecnológica aprovecha del conocimiento teórico científico producto de la investigación básica o sustantiva y organiza reglas técnicas cuya aplicación posibilita cambios en la realidad. (Sanchez Carlessi & Reyes, 2006)

El presente proyecto de tesis se utiliza el tipo de investigación tecnológica, debido a que con los conocimientos teóricos obtenidos mediante la investigación de la herramienta badboy se aplica en un entorno real mediante la implementación para mejorar el proceso de pruebas funcionales.

3.3 Nivel de la investigación

> **Investigación descriptiva:** Tiene como objetivo la descripción de los fenómenos a investigar, tal como es y cómo se manifiesta en el momento (presente) de realizarse el estudio y utiliza la observación como método descriptivo, buscando especificar las propiedades importantes para medir y evaluar aspectos, dimensiones o componentes. Pueden ofrecer la posibilidad de predicciones, aunque rudimentarias.
>
> Se sitúa en el primer nivel de conocimiento científico. Se incluyen en esta modalidad gran variedad de estudios (estudios correlaciónales, de casos, de desarrollo, etc.). (Sanchez Carlessi & Reyes, 2006)

El presente proyecto de tesis, es de nivel descriptivo porque describe en detalle la problemática en la que se encontraba la consultora de software y los procedimientos que se llevaron a cabo para mejorarla.

3.4 Enfoque de investigación

> **Enfoque Cuantitativo**: El esquema metodológico asume el positivismo lógico, en tal sentido busca que la medición sea objetiva y controlada. Supone procedimientos cuantitativos de procesamiento de datos, hace uso de la estadística descriptiva y/o inferencial. Las inferencias son derivadas del análisis estadístico, que van más allá de los datos, es decir trasciende, explica y generaliza al trabajar sobre planteamientos hipotéticos deductivos. (Sanchez Carlessi & Reyes, 2006)

El presente proyecto de tesis utiliza el enfoque cuantitativo porque se expresa numéricamente los indicadores de las variables dependientes recogidas en el pre test y en el post test.

3.5 Población y muestra

3.5.1 Población

Se define población como:

> Población o universo, conjunto de todos los casos que concuerdan con determinadas especificaciones. Una deficiencia que se presenta en algunos trabajos de investigación es que no describen lo suficiente las características de la población o consideran que la muestra la representa de manera automática.
>
> (Hernandez Sampieri, Fernadez Collado, & Baptista Lucio, 2014, pág. 174)

La población del presente proyecto comprende a las pruebas funcionales en una empresa consultora de software. A continuación, se presenta la población por cada variable independiente:

- ✓ Método de pruebas funcionales

  La población que se tomara es finita debido a que se conoce cantidad de errores por tipo de actualización.

- ✓ Automatizar

  La población que se tomara es finita debido a que se conoce el tiempo aproximado empleado en realizar las pruebas en la aplicación.

- ✓ Proceso

  La población que se tomara es finita debido a que se conoce el costo aproximado de horas al solucionar un fallo en la aplicación.

3.5.2 Muestra

Se define muestra como:

> Para el proceso cuantitativo, la muestra es un subgrupo de la población de interés sobre el cual se recolectarán datos, y que tiene que definirse y delimitarse de antemano con precisión, además de que debe ser representativo de la población. El investigador pretende que los resultados encontrados en la muestra se generalicen o extrapolen a la población.
>
> (Hernandez Sampieri, Fernadez Collado, & Baptista Lucio, 2014, pág. 173)

- ✓ Método de pruebas funcionales

  Se trabajó con una muestra del tipo no probabilística, debido a que se conoce la cantidad de errores por tipo de actualización.

  - Pre Test
    - Regularización de actualización: 5
    - Nuevo Requerimiento: 3
    - Reparación de defecto: 3

  - Post Test
    - Regularización de actualización: 2
    - Nuevo Requerimiento: 1
    - Reparación de defecto: 0

- ✓ Automatizar

  Se trabajó con una muestra del tipo no probabilístico, debido a que se observó el tiempo promedio que se toma en realizar las pruebas una vez terminado el despliegue en los ambientes de desarrollo

- Pre Test
    - Regularización de actualización: 10 h
    - Nuevo Requerimiento: 6h
    - Reparación de defecto: 5h

- Post Test
    - Regularización de actualización: 0.5h
    - Nuevo Requerimiento: 0.75h
    - Reparación de defecto: 0 h

✓ Proceso

Se trabajó con una muestra del tipo no probabilístico, debido a que se realizó el cálculo del costo total de las horas invertidas en corregir un fallo en la aplicación.

- Pre Test
    - Costo tiempo corrección: $214
    - Costo tiempo en pruebas: $97.5
    - Costo tiempo despliegue: $97.5

- Post Test
    - Costo tiempo corrección: $58.5
    - Costo tiempo en pruebas: $0.75
    - Costo tiempo despliegue: $0

3.6 Técnicas e instrumentos para la recolección de datos

3.6.1 Técnicas

Base de datos

"Se define una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular." (Perez Valdez, 2007)

Observación directa

"Constituye un proceso de atención, recopilación, selección y registro de información, para el cual el investigador se apoya en sus sentidos.

La ventaja de esta técnica es que permite obtener información independientemente que las personas estudiadas tengan que proporcionarla." (Hurtado, pág. 449)

Para el presente proyecto las técnicas que se utilizaron para cada variable dependiente se detallan en la tabla 03:

Tabla 03:
Técnicas

Variable Dependiente	Técnica
Identificación de fallos de software	Base de datos
Tiempo empleado en las pruebas funcionales	Base de datos
Costo en solucionar una falla en el software	Observación directa

Fuente: Elaboración propia

3.6.2 Instrumentos

Se define como instrumento al “Recurso que utiliza el investigador para registrar información o datos sobre las variables que tiene en mente” (Zorrilla, 1993, pág. 200)

Para el presente proyecto los instrumentos que se utilizaron para cada variable dependiente se detallan en la tabla 04:

Tabla 04:
Instrumentos

Variable Dependiente	Instrumento
Identificación de fallos de software	Software de gestión de tickets (Jira)
Tiempo empleado en las pruebas funcionales	Software de gestión de tickets (Jira)
Costo en solucionar una falla en el software	Documento de propuesta económica

Fuente: Elaboración propia

Confiabilidad de instrumentos

Según (Gomez) en su libro Metodología de la investigación científica define la confiabilidad como:

> Un instrumento de medición se refiere al grado en que su aplicación repetida al mismo sujeto u objeto produce resultados iguales.
>
> Por ejemplo, si la variable que deseamos medir fuera si la temperatura ambiental, nuestro instrumento de recolección de datos sería un termómetro.

Si en un momento dado éste indicara que hay 25º C, y un minuto más tarde se consultara otra vez y el mismo termómetro e indicara que hay 20º C, dicho termómetro no sería confiable, ya que su aplicación repetida produce resultados distintos.

La confiabilidad se obtiene realizando "pruebas piloto", antes de comenzar la recolección definitiva de los datos, y verificando que el instrumento permita obtener resultados similares en condiciones similares.

Según la página de Atlassian, el nivel de confiabilidad es avalada por el cuadrante mágico de Gartner en el año 2018. El software JIRA pertenece a Atlassian y es una herramienta para el registro y seguimiento de proyecto o incidencias (ver figura 24), como se puede observar Atlassian se encuentra en el cuadrante de líderes.

Figura 24: Cuadrante de Gartner
Elaboración: Atlassian

3.7 Técnicas para el procesamiento y análisis de los Datos

Ver Tabla 05

Tabla 05:
Matriz de Análisis de datos

Variable	Indicador	Escala de medición	Estadísticos descriptivos	Análisis inferencial
Identificación de fallos de software	Cantidad de errores por tipo de actualización	Escala de proporción/ razón	Tendencia central (media) Dispersión (desviación estándar)	U de Mann Whitney
Tiempo empleado en las pruebas funcionales	Tiempo empleado al ejecutar pruebas funcionales	Escala de proporción/ razón	Tendencia central (media) Dispersión (desviación estándar)	Prueba paramétrica (T student de Muestras Independientes)
Costo en solucionar una falla en el software	Costo por hora del programador/ Release Manager	Escala de proporción/ razón	Tendencia central (media) Dispersión (desviación estándar)	U de Mann Whitney

Fuente: Elaboración propia

CAPÍTULO IV: RESULTADOS DE LA INVESTIGACIÓN

4.1 Presentación de resultados

Generalidades

Para obtener las estadísticas de datos se utilizó el software de registro de incidencias JIRA.

Actualmente, el motivo por el que se realiza una actualización en la aplicación web son por 3 razones:

- ✓ Reparación de defecto
- ✓ Nuevo requerimiento
- ✓ Regularización de actualización

Método de pruebas funcionales

Antecedentes

Una vez que el analista funcional dé el OK de validación en la PC local del desarrollador, Envía un correo al Release Manager para que realice el proceso de congelado de objetos y su posterior despliegue en los ambientes de desarrollo.

Para el congelado de objetos se utiliza las siguientes herramientas:

- ✓ Para versionar páginas .NET se utiliza el controlador de versiones Visual SourceSafe. Ver Figura 25

- ✓ Para versionar objetos de BD ya sea procedimientos, funciones, triggers se utiliza la aplicación OSIRIS software que fue elaborado por la propia consultora. Ver Figura 26.

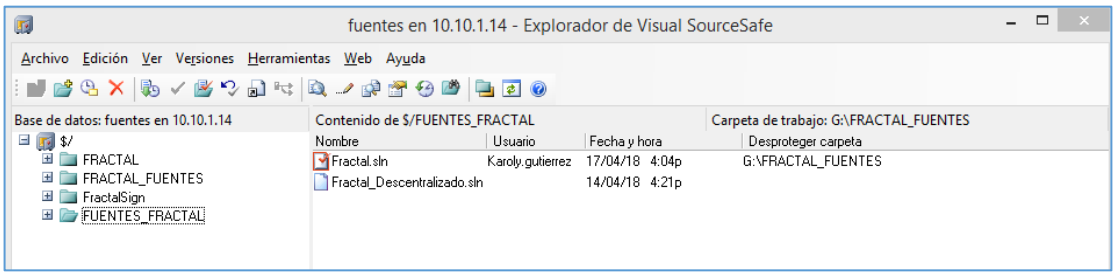

Figura 25: Controlador de versiones SourceSafe
Elaboración: propia

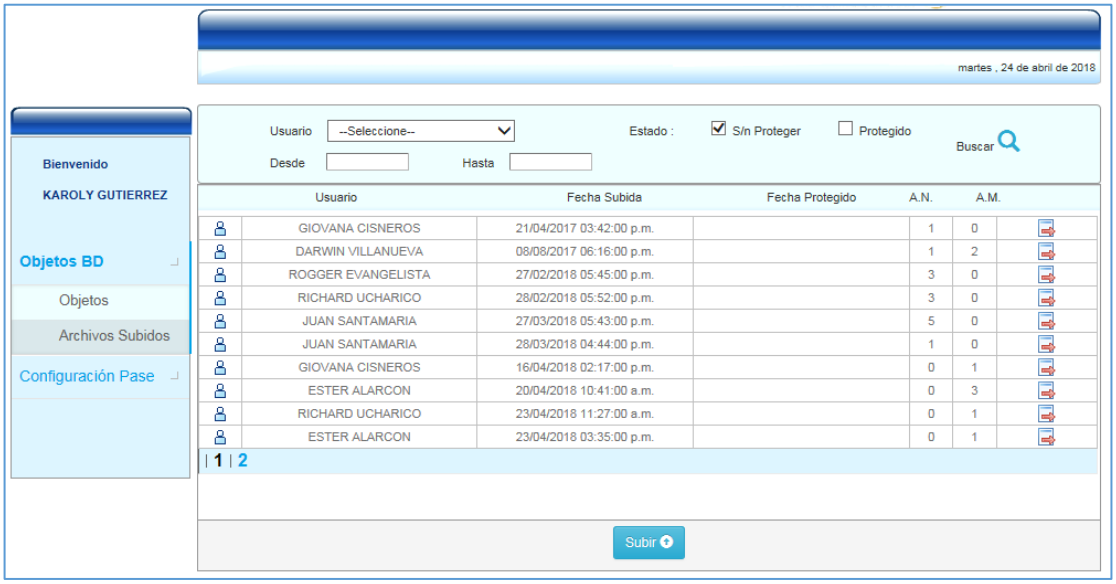

Figura 26: Aplicación Web Osiris
Elaboración: propia

Pre- Test

- ✓ El Release manager realiza el congelado de objetos (Scripts y paginas .NET) y luego de ello realiza el despliegue de páginas .NET y scripts en BD. Ver Figura 27.

- ✓ Una vez realizado el despliegue, el release manager envía un correo al analista funcional para que pueda realizar sus pruebas.

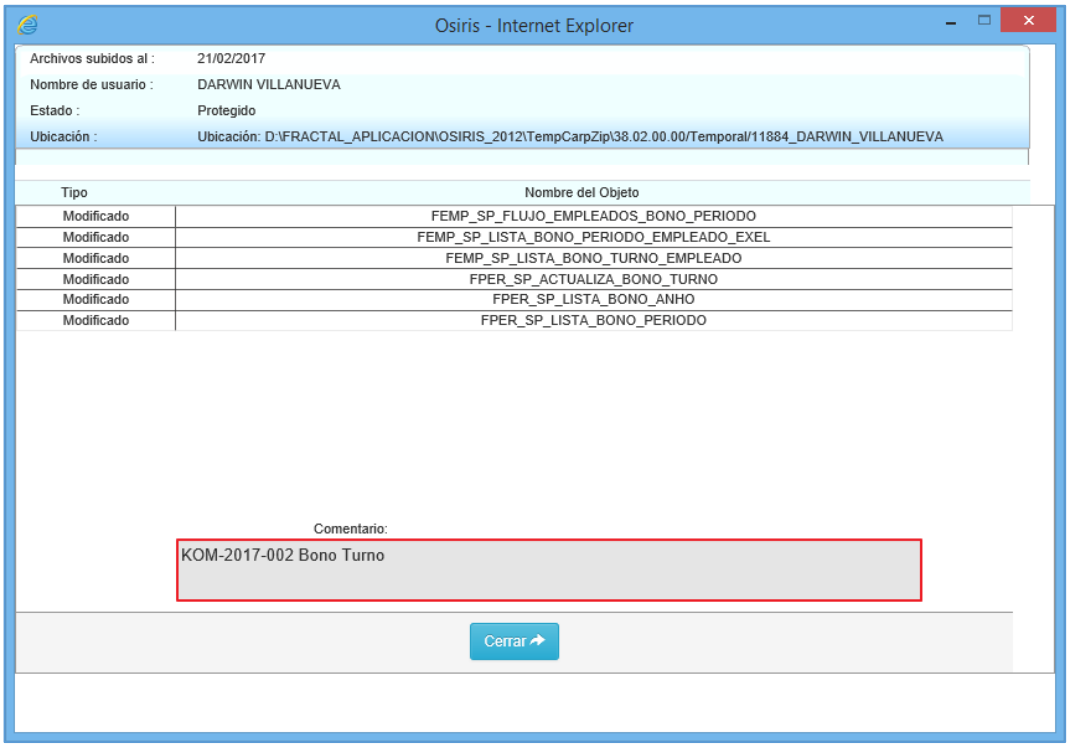

Figura 27: Ejemplo de objetos de BD a congelar
Elaboración: propia

El analista funcional utilizaba únicamente el método de caja negra para realizar sus pruebas funcionales, es decir realizaba sus pruebas en base a las funcionalidades de la aplicación las cuales fueron agregadas o modificadas.

Adicionalmente a ello, otro aspecto de mayor relevancia en el congelado de objetos es la BD ya que no hay forma de validar si los desarrolladores subieron o enviaron los scripts completos o correctos ya que los despliegues se realizan sobre la BD que ha sido manipulada por el desarrollador.

Post -Test

Se estableció un estándar que, el analista funcional aparte de utilizar las pruebas de caja negra también debe utilizar las pruebas de caja blanca tanto para código en .NET como BD (procedimientos almacenados, funciones. triggers, Vistas)

Sommerville define este tipo de pruebas como:

“Una aproximación al diseño de casos de prueba en donde las pruebas se derivan a partir del conocimiento de la estructura e implementación del software”. (Sommerville, 2005, pág. 509).

Esto evita a que el desarrollador coloque “código basura” en el caso de .NET o comandos que no sean, en caso de BD, compatibles con la versión del SQL Server 2008, esto debido a que muchos de los clientes cuentan con esta versión instalada en sus servidores.

Adicionalmente a lo detallado anteriormente, cuando se realice el despliegue en los ambientes de desarrollo previamente se debe realizar lo siguiente:

- ✓ El Release Manager debe generar un backup de producción y traerlo a los ambientes locales. En caso de no contar con accesos a los servidores del cliente se deberá coordinar el recojo del backup.

- ✓ Una vez en los ambientes locales se debe proceder con el restore del backup

- ✓ Luego de lo indicado anteriormente, recién se realiza el despliegue del desarrollo realizado

Una vez realizado lo mencionado el Release Manager envía un correo al analista funcional indicando que el ambiente se encuentra actualizado.

Con este estándar, se buscó minimizar errores con respecto a BD y .NET.

Automatizar pruebas funcionales

Pre –Test

Las pruebas funcionales se realizaban de forma manual.

Para el caso de:

- ✓ **Reparación de Defecto**

  El analista funcional solo realizaba la validación de la funcionalidad corregida.

- ✓ **Nuevo requerimiento**

  El analista funcional solo realizaba la validación de la funcionalidad modificada y/o agregada.

- ✓ **Regularización de Actualización**

  El área de CAC y/o analista funcional realizaba la validación de todas las funcionalidades de la aplicación.

Post –Test

Según se indicó en la situación pre-test, que solo en el caso de realizar una actualización por regulación de versión se realiza las pruebas en forma completa, ello debido al tiempo que se invierte en la realización de pruebas.

Según indica Stefany Rodríguez en una publicación que:

"La automatización de prueba es ventajosa en situaciones en las cuales el software se modifica constantemente, dado que hasta las modificaciones menores pueden ocasionar que funcionalidad ya desarrollada deje de funcionar".

Partiendo de la premisa que indica Stefany Rodríguez y debido a que la aplicación web se modifica constantemente por nuevos requerimientos, mejoras y /o correcciones se utilizó la herramienta Badboy para automatizar las pruebas funcionales.

En la página web (IT - mentor, pág. 9) indica también que unas las razones para automatizar es cuando el proceso de pruebas manual es muy largo y debido a que la aplicación web en estudio contiene múltiples módulos se consideró automatizar algunos test case con la ayuda del analista funcional.

Pero debido a la falta de tiempo y debido a que la aplicación a la que se implementó cuenta con varios módulos se realizó la automatización de un solo modulo, para ello se eligió el módulo más crítico que es el de procesos de pagos. Ver figura 28

Por ello, cuando se realiza el despliegue en los ambientes de desarrollo, se ejecuta la herramienta Badboy y esta realiza el proceso que previamente ha sido automatizado.

Al término de la ejecución la herramienta genera un reporte con los posibles errores que se pudo haber encontrado además de guardar un pantallazo del error.

Con la implementación de la herramienta mencionada se redujo el tiempo que se invertía en realizar las validaciones manualmente, además de errores que pudieron haber salido en producción pero que se corrigió gracias a que se detectó en la fase de pruebas.

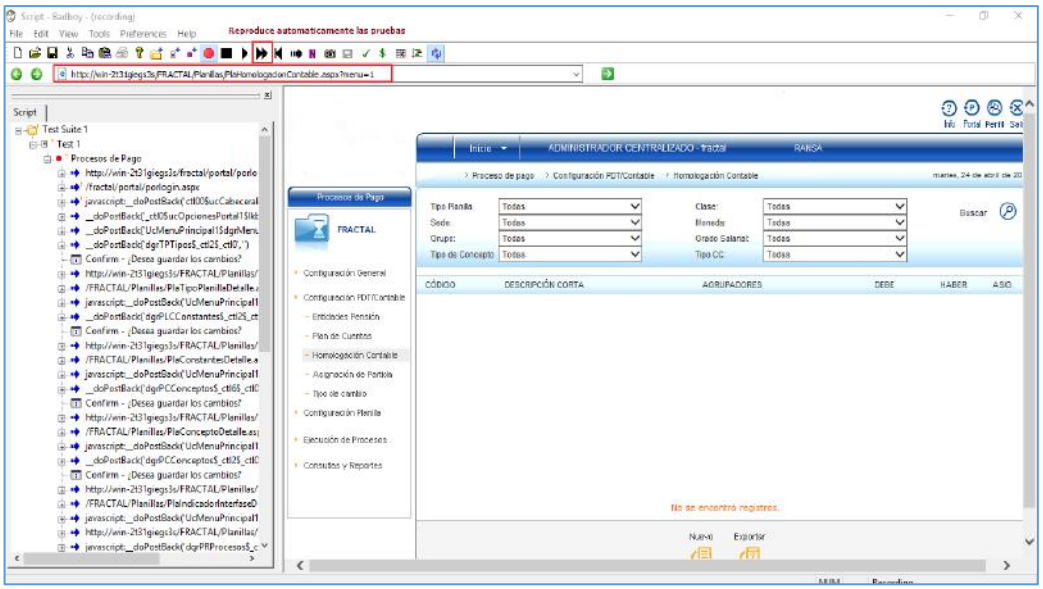

Figura 28: Herramienta badboy con la grabación de las pruebas del módulo de Procesos de pagos
Elaboración: propia

Proceso de trabajo entre el desarrollo y las pruebas funcionales

Pre – Test

✓ Regularización de Actualización

- El Release Manager informa al área Centro de Atención al Cliente (CAC) acerca de los clientes cuyos ambientes se encuentran desactualizados aproximadamente 6 meses a más.
- El área de CAC coordina con el cliente una posible fecha de actualización y envía un correo al Release Manager con la fecha.
- El Release manager, prepara los archivos de actualización y los despliega en el ambiente de desarrollo. y envía un correo al área de CAC para que se realice la validación de todas las funcionalidades en general
- El área de CAC realiza la validación respectiva de las funcionalidades.
    - En caso de encontrar errores, solicita al Analista funcional la corrección del mismo.

- De no haber errores, manda un correo al Release manager indicando conformidad de actualización.

✓ **Nuevo requerimiento**

- El Jefe de Proyecto se reúne con el cliente para definir los Requerimientos.
- El Jefe de proyecto se reúne con el analista funcional para indicarles los puntos del nuevo RQ a desarrollar el cual se detalla en el documento de especificación de requerimientos.
- Al analista funcional asigna el caso al desarrollador.
- El desarrollador realiza la tarea asignada y envía al Release Manager los objetos modificados (Scripts de SQL o Paginas .NET).
- El Release manager realiza el respectivo congelado de los objetos y realiza el despliegue en el ambiente de desarrollador. Luego de ello envía un correo al Analista funcional para la respectiva validación.
- El analista funcional realiza la validación de la funcionalidad modificada.

  En caso de encontrar errores solicita al desarrollador la corrección del mismo.

  De no haber errores solicita al Release manager el despliegue en Producción.

✓ **Reparación de Defecto**

- El área de Centro de Atención al cliente (CAC) reporta el error al analista funcional.
- Al analista funcional asigna el caso al desarrollador.
- El desarrollador realiza la tarea asignada y envía al Release Manager los objetos modificados (Scripts de SQL o Paginas .NET).
- El Release manager realiza el respectivo congelado de los objetos y realiza el despliegue en el ambiente de desarrollador. Luego de ello envía un correo al Analista funcional para la respectiva validación.

- El analista funcional realiza la validación de la funcionalidad modificada.
  En caso de encontrar errores solicita al desarrollador la corrección del mismo.
  De no haber errores solicita al Release manager el despliegue en Producción.

En los 3 casos mencionados anteriormente, para los casos 2 y 3 el analista funcional solo realizaba las pruebas en la funcionalidad donde se realizó el cambio, esto conllevaba a que al realizar el cambio algunas veces afectaba otra funcionalidad que el analista funcional no llegaba a validar, muchas veces por la falta de tiempo por la misma presión del usuario para realizar la entrega del desarrollo.

Para el 1er caso se realiza el inconveniente es el tiempo empleado en realizar las pruebas en toda la aplicación.

En todos los casos mencionados, una vez que se reportaba un error en el software se asignaba el caso a un desarrollador, lo que generaba que se detuviera la actividad que estaba realizando ese momento (muchas veces estaba desarrollando un nuevo requerimiento) para que pudiera resolver el caso sumado a eso también afectaba el tiempo del analista funcional el cual debía realizar las pruebas respectivas y el tiempo del release manager para realizar el despliegue en los ambientes de producción. Todo eso implicaba costos de tiempos perdidos.

Post –Test

A continuación, en la tabla 06 se identificó las etapas involucradas del ciclo de vida de software según el tipo o motivo de actualización según Somerville:

Tabla 06:
Etapa de ciclo de vida involucrado en tipo o motivo de actualización

Motivo Actualización / Etapa	Regularización de Actualización	Nuevo Requerimiento	Reparación de Defecto
Análisis y definición de RQ		X	
Diseño del sistema y del Software	X	X	X
Implementación y prueba de unidades	X	X	X
Integración y pruebas del sistema	X	X	X
Funcionamiento y mantenimiento	X	X	X

Fuente: Elaboración propia

Kendall indica lo siguiente:

> Antes de poner el sistema en funcionamiento es necesario probarlo. Es mucho menos costoso encontrar los problemas antes que el sistema se entregue a los usuarios.
>
> Una parte de las pruebas las realizan los programadores solos, y otra la llevan a cabo de manera conjunta con los analistas de sistemas.

Partiendo de la premisa de Kendall, se estableció el siguiente estándar:

- Durante la etapa de integración y pruebas a los sistemas, primero el desarrollador realizará sus pruebas relativas al desarrollo realizado en su computadora.
- El analista funcional, revisa el código y realiza las pruebas también desde su PC.
- Una vez realizado el despliegue en los ambientes de desarrollo, el desarrollador juntamente con el analista funcional realiza las pruebas finales.
- El release manager corre los scripts de pruebas grabadas en la herramienta badboy para verificar el correcto funcionamiento del módulo procesos de pago.

Sin embargo, con el compromiso por parte de los desarrolladores de realizar sus validaciones aún persisten errores en producción, los cuales son mínimos, pero no de mucha relevancia.

4.2 Análisis de resultados

Situación Pre- Test

✓ Identificación de fallos de Software

En la tabla 07 se muestra los datos obtenidos con respecto a la cantidad de actualizaciones que se realizaron por motivo.

Tabla 07:
Cantidad de actualizaciones por motivo de actualización. Muestra Pre-Test

Mes / Motivo Actualización	Regularización de Actualización	Nuevo Requerimiento	Reparación de Defecto
Enero		1	0
Febrero	3	0	2
Marzo	2	1	2
Abril – D10	0	1	3

Fuente: Elaboración propia

En la tabla 08 muestra los datos obtenidos con respecto a la cantidad de errores que reportó el cliente en la actualización que se aplicó al software. Algunos clientes reportan la incidencia luego de 1 o 2 meses después de realizada la actualización, esto debido a que la funcionalidad no es usada muy a menudo.

Tabla 08:
cantidad de errores encontrados por tipo de actualización. Muestra Pre-Test

Mes / Motivo Actualización	Marzo	Abril -D10
Regularización de Actualización	3	2 *
Nuevo Requerimiento	2	1
Reparación de defecto	1	2

(*) Aplicación cliente actualizado en el mes de febrero
Fuente: Elaboración propia

En la tabla 09 se muestra una fusión de datos de los meses de marzo y abril para realizar los respectivos análisis.

Tabla 09:
cantidad de errores por tipo de actualización Marzo –Abril. Muestra Pre-Test

Mes / Tiempo	Regularización de Actualización	Nuevo Requerimiento	Reparación de defecto
Marzo –Abril D10	5	3	3

Fuente: Elaboración propia

Análisis Descriptivo

En base a los datos obtenidos de la cantidad de actualizaciones de la aplicación por motivo se elaboró las estadísticas descriptivas para observar su comportamiento (ver tabla 10)

Tabla 10:
Estadística Descriptiva - Cantidad de errores por actualización. Pre-Test

Estadístico	Valor
Media	3,67
Desviación estándar	1,155
Mediana	3,0
Asimetría	1,732

Fuente SPSS

Prueba de Normalidad

Para la prueba de normalidad se plantea las siguientes Hipótesis:

- ✓ H_0 los datos siguen una distribución normal
- ✓ H_1 los datos no siguen una distribución normal

Con un nivel de significancia: $\alpha = 0.05$

Mediante el uso del software SPSS, se aplicó la prueba de normalidad Shapiro-wilk por que la muestra de los datos en menor a 50 ($n < 50$)

Tabla 11:
Prueba de normalidad de cantidad de errores por motivo de actualización. Pre-Test

Prueba de Normalidad – Shapiro - Wilk

	Estadístico	GI	Sig.
N° actualizaciones	,750	3	,000

Fuente: SPSS

Según la tabla 11, el grado de significancia es < 0,05 por lo tanto se acepta la H_1 concluyendo que los datos de la muestra no siguen una distribución normal.

✓ **Tiempo empleado en las pruebas funcionales**

Con la información obtenida se procedió con la interpretación de los datos obtenidos. Por lo que a continuación se detalla lo siguiente:

- En la tabla 12 se detalla el tiempo aproximado en realizar pruebas funcionales por tipo de actualización. Cabe indicar que este tiempo es por 1 sola actualización.

  En la presente tesis se denomina pruebas integrales al tipo de pruebas realizadas en su totalidad a toda la aplicación y pruebas unitarias es el tipo de pruebas que se realiza a la funcionalidad en específico que fue modificada o corregida.
- En la tabla 13 se detalla el tiempo aproximado invertido por mes en la realización de pruebas funcionales por tipo de actualización.

Tabla 12:
Tiempo aprox. por unidad

Tipo actualización	Tiempo	Tipo de prueba
Reg. de actualización	5h	Pruebas integrales
Nuevo RQ	3h	Pruebas unitarias
Rep. Por defecto	1h	Pruebas unitarias

Fuente: Elaboración propia

Tabla 13:
Tiempo aproximado invertido al mes por tipo de actualización en pruebas. Pre-Test

Mes	Regularización de Actualización	Nuevo Requerimiento	Reparación de Defecto
Enero	5h	3h	0h
Febrero	15h	0h	2h
Marzo	10h	3h	2h
Abril – D10	0h	3h	3h

Fuente: Elaboración propia

En la tabla 14 se muestra una fusión de datos de los meses de marzo y abril para realizar los respectivos análisis.

Tabla 14:
Tiempo invertido en pruebas Marzo –Abril. Muestra Pre-Test

Mes/ Tiempo	Regularización de Actualización	Nuevo Requerimiento	Reparación de defecto
Marzo –Abril D10	10h	6h	5h

Fuente: Elaboración propia

Análisis Descriptivo

En base al tiempo empleado en pruebas por motivo de actualización el análisis descriptivo (ver tabla 15)

Tabla 15:
Estadística Descriptiva – Tiempo invertido en pruebas. Pre-Test

Estadístico	Valor
Media	6
Desviación estándar	3,606
Mediana	5
Asimetría	1,152

Fuente: SPSS

Prueba de Normalidad

Para la prueba de normalidad se plantea las siguientes Hipótesis:

- ✓ H_0 los datos siguen una distribución normal
- ✓ H_1 los datos no siguen una distribución normal

Con un nivel de significancia: $\alpha = 0.05$

Mediante el uso del software SPSS, se aplicó la prueba de normalidad Shapiro-wilk por que la muestra de los datos en menor a 50 ($n < 50$)

Tabla 16:
Prueba de normalidad de tiempo invertido en realizar pruebas funcionales. Pre-test

	Prueba de Normalidad – Shapiro - Wilk		
	Estadístico	GI	Sig.
Costo ($)	,942	3	,537

Fuente: SPSS

Según la tabla 16, el grado de significancia es > 0,05 por lo tanto se acepta la H_0 concluyendo que los datos de la muestra siguen una distribución normal por lo tanto son paramétricos.

✓ **Costo en solucionar una falla en el software**

Con la información obtenida se procedió con la interpretación de los datos obtenidos respecto al costo que implico en corregir una falla o error en la aplicación.

En la tabla 17 se detalla que adicionalmente al costo de las horas que implica realizar la corrección en si del error también se considera el costo por las horas invertidas en realizar pruebas y el costo de las horas invertidas en realizar el despliegue en producción como un costo total en solucionar una falla en el software.

Tabla 17:
Costo Total por mes 2018. Pre-Test

Mes	Costo Tiempo Corrección ($)	Costo Tiempo Pruebas ($)	Costo Tiempo Despliegue ($)	Costo Total ($)
Enero	0	0	0	
Febrero	78	39	39	156
Marzo	117	39	58.50	214
Abril – D10	97.5	58.50	39	195

Fuente: Elaboración propia

En la tabla 18 se muestra una fusión de datos de los meses de marzo y abril para realizar los respectivos análisis.

Tabla 18:
Costo Marzo–Abril. Muestra Pre-Test

Mes	Costo Tiempo Corrección ($)	Costo Tiempo Pruebas ($)	Costo Tiempo Despliegue ($)	Costo Total ($)
Marzo –Abril D10	214	97.50	97.5	409

Fuente: Elaboración propia

Análisis Descriptivo

En base a los costos en solucionar un defecto de software se elaboró su análisis descriptivo (ver tabla 19)

Tabla 19:
Estadística Descriptiva – Costo en solucionar defecto de software. Pre-Test

Estadístico	Valor
Media	136,3333
Desviación estándar	67,26131
Mediana	97,5000
Asimetría	1,732

Fuente: SPSS

Prueba de Normalidad

Para la prueba de normalidad se plantea las siguientes Hipótesis:

- ✓ H_0 los datos siguen una distribución normal
- ✓ H_1 los datos no siguen una distribución normal

Con un nivel de significancia: $\alpha = 0.05$

Mediante el uso del software SPSS, se aplicó la prueba de normalidad Shapiro-wilk por que la muestra de los datos en menor a 50 ($n < 50$)

Tabla 20:
Prueba de normalidad de Costo invertido en solucionar fallas en el software. Pre-Test

Prueba de Normalidad – Shapiro - Wilk			
	Estadístico	GI	Sig.
Costo ($)	,750	3	,000

Fuente: SPSS

Según la tabla 20, el grado de significancia es <0,05 por lo tanto se acepta la H_1 concluyendo que los datos de la muestra no siguen una distribución normal.

Situación Post- Test

El periodo post-test tuvo una duración de (25) días para el análisis de los resultados de la implementación de la herramienta badboy. El post test fue del 11 de abril hasta el 05 de mayo del 2018.

✓ Identificación de fallos de Software

Después de la implementación de la herramienta se volvió a recolectar información sobre la cantidad de actualizaciones realizadas (ver tabla 21) y la cantidad de errores encontrados en cada actualización (ver tabla 22).

Tabla 21:
Cantidad de actualizaciones realizadas. Muestra Post-Test

Motivo Actualización	N° Actualizaciones
Reg. de Actualización	2
Nuevo RQ	3
Reparación de Defecto	0

Fuente: Elaboración propia

Tabla 22:
Cantidad de errores encontrados. Muestra Post-Test

Motivo Actualización	N° Errores
Reg. de Actualización	2
Nuevo RQ	1
Reparación de Defecto	0

Fuente: Elaboración propia

Análisis Descriptivo

En base a los datos de cantidad de errores por motivo de actualización se realizó su análisis descriptivo (ver tabla 23)

Tabla 23:
Estadística Descriptiva – Cantidad de errores Post-Test

Estadístico	Valor
Media	1,00
Desviación estándar	1,000
Mediana	1,00
Asimetría	,000

Fuente: SPSS

Prueba de Normalidad

Para la prueba de normalidad se plantea las siguientes Hipótesis:

- ✓ H_0 los datos siguen una distribución normal
- ✓ H_1 los datos no siguen una distribución normal

Con un nivel de significancia: $\alpha = 0.05$

Mediante el uso del software SPSS, se aplicó la prueba de normalidad Shapiro-wilk por que la muestra de los datos en menor a 50 (n < 50)

Tabla 24:
Prueba de normalidad de cantidad de errores- Post-Test

Prueba de Normalidad – Shapiro - Wilk

	Estadístico	GI	Sig.
N° de Actualizaciones	1,000	3	1,000

Fuente: SPSS

Según la tabla 24, el grado de significancia es > 0,05 por lo tanto se acepta la H_0 concluyendo que los datos de la muestra siguen una distribución normal.

- ✓ **Tiempo empleado en las pruebas funcionales**

En la tabla 25 se presenta los datos recolectados después de la implementación en cuanto a los tiempos empleados en las pruebas funcionales por motivo de actualización.

Tabla 25:
Tiempo aproximado invertido por tipo de actualización. Muestra Post-Test

Mes	Regularización de Actualización	Nuevo Requerimiento	Reparación de Defecto	Total Tiempo
Abril -Mayo	0,5h	0,75h	0	1,25h

Fuente: Elaboración propia

Análisis Descriptivo

En base a los datos de tiempo invertido en pruebas por motivo de actualización se realizó el análisis descriptivo (ver tabla 26)

Tabla 26:
Estadística Descriptiva – tiempo invertido en pruebas. Post-Test

Estadístico	Valor
Media	0,2667
Desviación estándar	,41932
Mediana	0,0500
Asimetría	1,704

Fuente: SPSS

Prueba de Normalidad

Para la prueba de normalidad se plantea las siguientes Hipótesis:

- ✓ H_0 los datos siguen una distribución normal
- ✓ H_1 los datos no siguen una distribución normal

Con un nivel de significancia: $\alpha = 0.05$

Mediante el uso del software SPSS, se aplicó la prueba de normalidad Shapiro-wilk por que la muestra de los datos en menor a 50 ($n < 50$)

Tabla 27:
Prueba de normalidad de tiempo invertido en realizar pruebas funcionales-– Post Test

	Prueba de Normalidad – Shapiro - Wilk		
	Estadístico	GI	Sig.
Tiempo invertido en pruebas funcionales	,800	3	,114

Fuente: SPSS

Según la tabla 27, el grado de significancia es >0,05 por lo tanto se acepta la H_0 concluyendo que los datos de la muestra siguen una distribución normal por lo tanto son paramétricos.

✓ **Costo en solucionar una falla en el software**

En la tabla 28 se presentan los datos recolectados después de la implementación en cuanto al costo involucrados para solucionar una falla en el software.

Tabla 28:
Costo en solucionar una falla en el software – Post-test

Mes	Costo Tiempo Corrección($)	Costo Tiempo Pruebas ($)	Costo Tiempo despliegue ($)	Costo Total ($)
Abril-Mayo	58.5	0.75	0*	59.25

(*) No se considera costo de despliegue porque solo se corrigió el error más aún no se coordina el despliegue en producción.
Fuente: Elaboración propia

Análisis Descriptivo

En base a los datos de los costos involucrados para solucionar una falla en el software se realizó su análisis descriptivo (ver tabla 29)

Tabla 29:
Estadística Descriptiva- Costo en solucionar defecto de software. Post-Test

Estadístico	Valor
Media	19,7500
Desviación estándar	33,56058
Mediana	0,7500
Asimetría	1,731

Fuente: SPSS

Prueba de Normalidad

Para la prueba de normalidad se plantea las siguientes Hipótesis:

- ✓ H_0 los datos siguen una distribución normal
- ✓ los datos no siguen una distribución normal

Con un nivel de significancia: α = 0.05

Mediante el uso del software SPSS, se aplicó la prueba de normalidad Shapiro-wilk por que la muestra de los datos en menor a 50 (n < 50)

Tabla 30:
Prueba de normalidad de Costo invertido en solucionar fallas en el software. Post-Test

Prueba de Normalidad – Shapiro - Wilk

	Estadístico	GI	Sig.
Costo ($)	,760	3	,021

Fuente: SPSS

Según la tabla 30, el grado de significancia es < 0,05 por lo tanto se acepta la H_1 concluyendo que los datos de la muestra no siguen una distribución normal por lo tanto no son paramétricos.

- ✓ **Contrastación de Hipótesis**

  - **Hipótesis Especifica 01**

    Si se describe el método de pruebas funcionales del analista funcional, entonces se identificarán los fallos en el desarrollo de software.

    H_0 No existe diferencia significativa entre las muestras de la cantidad de errores por tipo de actualización en el Pre Test y las muestras de cantidad de errores por tipo de actualización en el Post Test.

    H_1: Si existe diferencia significativa entre las muestras de la cantidad de errores por tipo de actualización en el Pre Test y las muestras de cantidad de errores por tipo de actualización en el Post Test.

    Por lo tanto, tenemos:

    H_0 Si se describe el método de pruebas funcionales del analista funcional, entonces NO se identificarán los fallos en desarrollo de software.

    H_1: Si se describe el método de pruebas funcionales del analista funcional, entonces se identificarán los fallos en desarrollo de software.

    Definimos el nivel de significancia α = 0.05

    - Si sig. o p-valor ≥ 0.05 → se acepta la H_0
    - Si sig. o p-valor < 0.05 → se acepta la H_1

    Grupo de muestra: Independiente

    - Pre-test → no normal
    - Post-test → normal

    Estadístico de prueba: U de Mann Whitney

Realizando la prueba de hipótesis, se obtienen los resultados se obtienen los resultados mostrados en la figura 29.

Estadísticos de prueba[a]

	Muestras
U de Mann-Whitney	,000
W de Wilcoxon	6,000
Z	-1,993
Sig. asintótica (bilateral)	,046
Significación exacta [2* (sig. unilateral)]	,100[b]

a. Variable de agrupación: TipoPrueba

b. No corregido para empates.

Figura 29: Contrastación de primera hipótesis
Fuente: SPSS

Lo que nos indica que 0,046 < 0.05 por lo tanto se acepta H_1

H_1: Si se describe el método de pruebas funcionales del analista funcional, entonces se identificarán los fallos en desarrollo de software.

Por lo que concluimos que se debe aplicar la Variable Independiente: Describir el método de pruebas funcionales.

- **Hipótesis Especifica 02**

Si se automatiza las pruebas funcionales de aplicaciones Web, entonces se reducirá el tiempo empleado en las pruebas funcionales básicas de la regla de negocio.

H_0: No existe diferencia significativa entre las muestras del tiempo empleado en las pruebas funcionales en el Pre Test y las muestras del tiempo empleado en las pruebas funcionales en el Post Test.

H_1: Si existe diferencia significativa entre las muestras del tiempo empleado en las pruebas funcionales en el Pre Test y las muestras del tiempo empleado en las pruebas funcionales en el Post Test.

Por lo tanto, tenemos:

H_0 Si se automatiza las pruebas funcionales de aplicaciones web, entonces NO se reducirá el tiempo empleado en las pruebas funcionales básicas de las reglas de negocio.

H_1: Si se automatiza las pruebas funcionales de aplicaciones web, entonces se reducirá el tiempo empleado en las pruebas funcionales básicas de las reglas de negocio.

Definimos el nivel de significancia $\alpha = 0.05$

- Si sig. o p-valor ≥ 0.05 → se acepta la H_0
- Si sig. o p-valor < 0.05 → se acepta la H_1

Grupo de muestra: Independiente

- Pre-test → normal
- Post-test → normal

Estadístico de prueba: T de student de muestras independientes

Tabla 31:
Contrastación de segunda hipótesis

	Prueba de Levene de igualdad de varianzas		Prueba t para igualdad de medias						
	F	Sig.	t.	gl.	Sig. (bilateral)	Dif. de medias	Dif. de error Estándar	95% de intervalo de confianza de la diferencia	
								Inferior	Superior
Se asumen varianzas iguales	8,638	,042	4,266	4	,013	6,58333	1,54335	2,29829	10,86837
No se asumen varianzas iguales			4,266	2,083	,047	6,58333	1,54335	,19087	12,97580

Fuente: SPSS

En la tabla 31 se observa la prueba de Levene para determinar si existen varianzas iguales o diferentes. Por lo tanto, si se tiene Sig = ,042 el cual es menor que α = 0.05, entonces no se asumen varianzas iguales.

Al no tener varianzas iguales, se realiza la prueba de T student, donde el valor de Sig (bilateral) es 0,047 el cual es menor a 0.05 por lo tanto se acepta la H_1.

H_1: Si se automatiza las pruebas funcionales de aplicaciones web, entonces se reducirá el tiempo empleado en las pruebas funcionales básicas de las reglas de negocio.

Por lo que concluimos que se debe aplicar la Variable Independiente: Automatizar las pruebas funcionales.

- **Hipótesis Especifica 03**

Si se rediseña el proceso del trabajo entre el desarrollo y las pruebas funcionales de software, entonces se reducirán los costos en solucionar fallas en el software.

H_0: No existe diferencia significativa entre las muestras del costo en solucionar una falla en el software en el Pre Test y las muestras del costo en solucionar una falla en el software en el Post Test.

H_1: Si existe diferencia significativa entre las muestras del costo en solucionar una falla en el software en el Pre Test y las muestras del costo en solucionar una falla en el software en el Post Test.

Por lo tanto, tenemos:

H_0 Si se rediseña el proceso de trabajo entre el desarrollo y las pruebas funcionales de software, entonces NO se reducirán los costos en solucionar fallas en el software.

H_1: Si se rediseña el proceso de trabajo entre el desarrollo y las pruebas funcionales de software, entonces se reducirán los costos en solucionar fallas en el software.

Definimos el nivel de significancia α = 0.05

- Si sig. o p-valor ≥ 0.05 → se acepta la H_0
- Si sig. o p-valor < 0.05 → se acepta la H_1

Grupo de muestra: Independiente

- Pre-test → no normal
- Post-test → no normal

Estadístico de prueba: U de Mann Whitney

Realizando la prueba de hipótesis, se obtienen los resultados se obtienen los resultados mostrados en la figura 30.

Estadísticos de prueba[a]

	Muestras
U de Mann-Whitney	,000
W de Wilcoxon	6,000
Z	-1,993
Sig. asintótica (bilateral)	,046
Significación exacta [2* (sig. unilateral)]	,100[b]

a. Variable de agrupación: TipoPrueba

b. No corregido para empates.

Figura 30: Contrastación de tercera hipótesis
Fuente: SPSS

Lo que nos indica que 0,046 < 0.05 por lo tanto se acepta H_1

H_1: Si se rediseña el proceso de trabajo entre el desarrollo y las pruebas funcionales de software, entonces se reducirán los costos en solucionar fallas en el software.

Por lo que concluimos que se debe aplicar la Variable Independiente: Rediseñar el Proceso de trabajo entre el desarrollo y las pruebas funcionales

CONCLUSIONES

- Para mejorar las pruebas funcionales, se realizó la implementación de la herramienta Badboy para lo cual se identificaron los casos de prueba y se realizaron las grabaciones de las mismas con el fin de ejecutar estas tareas de manera desatendidas; se optó por esta herramienta sobre otras ya que este software resultó ser compatible con la aplicación web a automatizar.

- Para identificar los fallos en el desarrollo de software fue necesario describir el método que el analista funcional utilizaba para detectar fallas en un desarrollo para luego establecer un estándar de trabajo previo a un pase a producción tales como validacion de código en .Net , objetos de BD y recrear un ambiente similar al de producción con ello se observó que en el equipo de desarrollo evidenciaron mejoras en los entregables obteniéndose como resultados de un antes con un total de 11 a un después con un total 3 errores. Estos resultados están basados en una sumatoria total de actualizaciones realizadas en varios clientes.

- Para reducir el tiempo empleado en la realización de pruebas funcionales básicas se recurrió a la automatización solo del módulo de procesos de pago de los clientes a los cuales se les había programado las fechas de pases a producción. Se realizó la actualización de los ambientes de desarrollo, se corrió el script y se observó la reducción del esfuerzo invertido en la realización de pruebas de un total de 21 horas a 1h y 15 aprox. Lo que se ve la reducción considerable de tiempos que también se ve influenciado por la cantidad de actualizaciones programados en ese periodo de tiempo que fue menor al pre- test.

- Para reducir los costos en corregir errores en el desarrollo, se estableció un estándar de trabajo para el equipo de desarrollo y el analista funcional, los cuales realizan sus pruebas unitarias en modo local antes del congelamiento para luego solicitar el despliegue en los ambientes de desarrollo, con esta definición también se evita el retrabajo en cuanto a congelamiento innecesario.

Con esto se verificó que habiendo reducido los errores también se redujo el costo en solucionar errores de un total de $409 a $59.25.

RECOMENDACIONES

- La consultora de Software debe proseguir con la implementación en los demás módulos y/o funcionalidades para optimizar el tiempo en las pruebas funcionales y por consiguiente evitar errores en producción.

- Adicionar la herramienta JMeter el cual es compatible con Badboy ya que este en cada prueba grabada genera scripts lo cual se puede utilizar en JMeter para la realización de pruebas de stress a la aplicación web para mejorar el rendimiento de la misma.

- Se debe respetar y hacer cumplir los estándares establecidos en la presente investigación para que se sigan obteniendo los mismos resultados o mejorar los mismos.

- Para hacer uso óptimo de la herramienta, se solicitó realizar una modificación al código fuente de la aplicación web, ya que durante la ejecución del script de pruebas se verificó unos cuadros de diálogos lo cual dificulta la ejecución de manera desatendida de las pruebas.

FUENTES DE INFORMACIÓN

Bibliográficas

Campbell, D., & Stanley, J. (1996). *Experimental and quasi-experimental designs for research.* Chicago.

Gomez, M. (2016). *Metodologia de la investigacion Cientifica* (2da Edicion ed.). Buenos Aires, Argentina: Brujas.

Gonzalez, C. D. (1998). *Costos 1.* Mexico: Ecafsa.

Gran Diccionario de la Lengua Española. (2016). Larousse Editorial.

Hernandez Sampieri, Fernadez Collado, & Baptista Lucio. (2014). *Metodologia de la investigacion.*

Hurtado, J. (2000). *Metodologia de la investigacion Holistica* (3era edicion ed.). Caracas, Venezuela.

Kendall, K. E., & Kendall, J. E. (2005). *Analisis y diseño de Sistemas* (Sexta Edicion ed.). Mexico: Pearson Educacion.

Pressman, R. S. (2010). *Ingenieria de Software , Un Enfoque Practico* (7ma Edicion ed.). Mexico D.F: Mc Graw Hill.

Sanchez Carlessi, H., & Reyes, M. C. (2006). *Metodologia y diseño de la investigacion Cientifica.*

Sommerville, I. (2005). *Ingenieria de Software* (7ma Edicion ed.). Madrid: Pearson Educacion S.A.

Wong, H. (Diciembre de 2017). Automatizacion de Pruebas Funcionales de Software. Lima.

Zorrilla. (1993).

Electrónicas

Blasco, A. (10 de Enero de 2012). *Clavei*. Recuperado el 18 de Febrero de 2018, de https://www.clavei.es/blog/2012/automatizacion-de-pruebas-un-paso-fundamental-para-mejorar-la-calidad-del-software/

Chinarro Morales, E., Ruiz Rivera, M. E., & Ruiz Lizama, E. (2017). Obtenido de http://revistasinvestigacion.unmsm.edu.pe/index.php/idata/article/view/13498/11952

Esmite, I., Farias, M., Farias, N., & Perez, B. (s.f.). *Centro de Ensayos de Software.* Recuperado el 13 de Marzo de 2018, de http://www.ces.com.uy/documentos/imasd/CES-CACIC07-Automatizacion_y_Gestion_Pruebas_Funcionales.pdf

Cuba Montenegro, R. (s.f.). Obtenido de http://oa.upm.es/49320/1/PFC_RAFAEL_CUBAS_MONTENEGRO.pdf

Garcia Gonzalez, J. C., & Andre Ampuero, M. (2015). Obtenido de cinfo.idict.cu/index.php/cinfo/article/download/672/516

Gonzales Palacios, L. (2009). Metodo para generar casos de prueba funcional en el desarrollo de Software. *Revista Ingenierías Universidad de Medellín, 8*. Obtenido de http://www.redalyc.org/articulo.oa?id=75017199005

IT - mentor. (s.f.). *Pruebas de Software.*

Martinez Taleno, G. (2012). *https://repositoriotec.tec.ac.cr/*. Obtenido de RepositorioTEC: https://repositoriotec.tec.ac.cr/bitstream/handle/2238/4019/Estrategia%20operativa%20para%20pruebas%20de%20automatizaci%C3%B3n%20y%20rendimiento.pdf?sequence=1&isAllowed=y

Martinez, A. (01 de Diciembre de 2014). *Sociedad Informática del Gobierno Vasco.* Recuperado el Marzo de 2018, de http://www.ejie.eus/contratacion/-/contratacion-informacion-tecnica/

NTP-ISO/IEC 12207 . (13 de 07 de 2006). Obtenido de http://www.senasa.gob.pe/senasa/wp-content/uploads/2014/11/Certifican-citricos-a-mexico_26_mayo_2105_2.pdf

Oliveros, A., Danyans, F. J., & Mastropietro, M. L. (2014). Recuperado el 18 de Febrero de 2018, de https://www.researchgate.net/profile/Alejandro_Oliveros/publication/265755196_Stakeholders_en_los_requerimientos_de_aplicaciones_Web/links/541ac4bd0cf2218008bfdf5b/Stakeholders-en-los-requerimientos-de-aplicaciones-Web.pdf

Perez Valdez, D. (26 de Octubre de 2007). *Platzy*. Recuperado el 01 de Abril de 2018, de http://www.maestrosdelweb.com/que-son-las-bases-de-datos/

Polanco Paredes, K. D. (2014). *http://tesis.ucsm.edu.pe*. Obtenido de http://tesis.ucsm.edu.pe/repositorio/bitstream/handle/UCSM/4649/71.0523.IS.pdf?sequence=1&isAllowed=y

Rodriguez Johnson, S. G. (06 de Diciembre de 2016). Recuperado el 11 de Marzo de 2018, de https://es.linkedin.com/pulse/5-herramientas-para-la-automatizaci%C3%B3n-de-pruebas-rodriguez-johnson

Salazar Rodriguez, J. (2016). Obtenido de http://repository.udistrital.edu.co/bitstream/11349/5194/1/SalazarRodriguezJuanCamilo2016.pdf

ANEXOS

Anexo 01: Matriz de Consistencia

Tabla A01.1:
Matriz de Consistencia

Problemas General	Objetivos General	Hipótesis General	Variables Independiente	Indicador V.I.	Variables Dependiente	Indicador V.D.
¿Cómo mejorar las pruebas funcionales en aplicaciones web?	Implementar la herramienta Badboy para mejorar el entorno de pruebas funcionales en aplicaciones Web	Si se implementa la herramienta Badboy, entonces mejorara el entorno de pruebas funcionales en aplicaciones web	Herramienta Badboy		Pruebas funcionales en aplicaciones web	
Problemas Especifico	**Objetivos Específicos**	**Hipótesis Especificas**				
¿Cómo identificar fallos en el desarrollo de software?	Describir el método de pruebas funcionales del analista funcional para identificar fallos en el desarrollo de software	Si se describe el método de pruebas funcionales del analista funcional, entonces se identificarán los fallos en desarrollo de software	Método de pruebas funcionales	Si/No	Identificación de fallos de software	Cantidad de errores por tipo de actualización
¿Cómo reducir el tiempo en las pruebas funcionales básicas de las reglas de negocio?	Automatizar las pruebas funcionales de las aplicaciones web, para reducir el tiempo en las pruebas funcionales básicas de las reglas de negocio.	Si se automatiza las pruebas funcionales de aplicaciones web, entonces se reducirá el tiempo empleado en las pruebas funcionales básicas de las reglas de negocio	Automatizar pruebas funcionales	Si/No	Tiempo empleado en las pruebas funcionales	Tiempo empleado en ejecutar pruebas funcionales
¿Cómo reducir los costos en corregir una falla en el software?	Rediseñar el proceso de trabajo entre el desarrollo y las pruebas funcionales del software para reducir los costos en solucionar fallas en software.	Si se rediseña el proceso de trabajo entre el desarrollo y las pruebas funcionales de software, entonces se reducirán los costos en solucionar fallas en el software	Proceso de trabajo entre el desarrollo y las pruebas funcionales	Si/No	Costo en solucionar una falla en el software	Costo por Hora del programador/ Release Manager

Fuente: Elaboración propia

Anexo 02: Matriz de Operacionalización

Tabla A02.1:
Matriz de Operacionalización

Variable Independiente	Indicador	Definición Conceptual	Definición Operacional
Métodos de pruebas funcionales	Si/No	"Las pruebas funcionales son las que se aplican al producto final y permiten detectar en que puntos el producto no cumple sus especificaciones, el método de verificar su correcto funcionamiento es a partir de casos de pruebas funcionales y con herramientas para la documentación de pruebas funcionales." (Gonzales Palacios, 2009)	Procedimientos que utiliza el analista funcional para realizar la validación de un requerimiento.
Automatizar pruebas funcionales	Si/No	"La automatización consiste en la construcción de un conjunto de scripts reutilizables, con los que podemos aumentar drásticamente la capacidad de testear software. Permite reducir los costes de cualquier organización que necesite probar sucesivas versiones de un mismo producto." (Blasco, 2012)	Procedimiento mediante el cual se graban las pruebas unitarias e integrales mejorando la calidad del entregable.
Proceso de trabajo entre el desarrollo y las pruebas funcionales	Si/No	"Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, operación y el mantenimiento de un producto de software, que abarca toda la vida del sistema desde la definición de sus requisitos hasta la finalización de su uso. " (NTP-ISO/IEC 12207 , 2006)	Procedimiento mediante el cual el Requerimiento pasa desde el desarrollo hasta el despliegue en los ambientes de desarrollo y su posterior validación.
Variable Dependiente	**Indicador**	**Definición Conceptual**	**Definición Operacional**
Identificación de fallos de software	Cantidad de errores por tipo de actualización	"Aunque probar es tedioso, es una serie esencial de pasos que ayuda a asegurar la calidad del eventual sistema. Es mucho menos inquietante probar de antemano que tener un sistema probado deficientemente que falle después de la instalación" (Kendall & Kendall, 2005)	N° errores por actualización
Tiempo empleado en pruebas funcionales	Tiempo empleado al ejecutar pruebas funcionales	"Una parte de las pruebas las realizan los programadores solos y otra la llevan a cabo de manera conjunta con los analistas de sistemas." (Kendall & Kendall, 2005)	T.Total_pruebas = T.P_Integrales + T.P_Unitarias
Costo en solucionar una falla en el software	Costo por hora de programador	"Es mucho menos costoso encontrar los problemas antes que el sistema se entregue a los usuarios" (Kendall & Kendall, 2005)	C.Tot. = (C.T_Corregir+C.T_Pruebas + T_Despliegue) * CostoxHora

Fuente: Elaboración propia

Anexo 03: Implementación de la Herramienta BadBoy

El software que se implementó para el desarrollo de la automatización de las pruebas es BadBoy el cual es compatible para aplicaciones web que funcionan bajo el navegador Internet Explorer. Como se había mencionado anteriormente, dado que las funcionalidades de la aplicación web donde se trabajó la implementación solo funcionan bajo ese navegador esta herramienta resultó ser beneficioso.

Arquitectura Física de la Implementación

Para la arquitectura Física se consideró lo siguiente:

- Servidor de aplicaciones de los ambientes locales.

- El Equipo Físico que utiliza el Release Manager, para el cual se solicitó el aumento de memoria RAM a 12 GB. Figura 31.

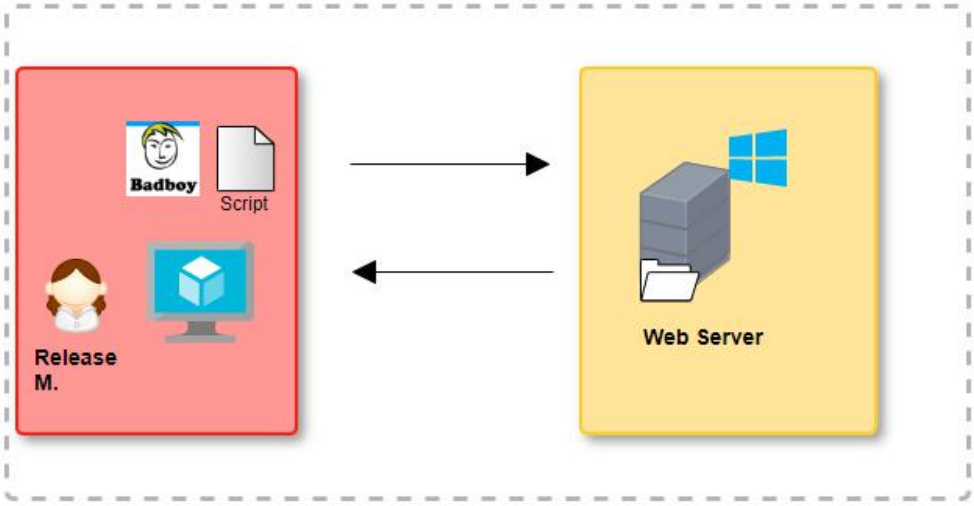

Figura 31: Arquitectura Física de la solución
Fuente: Elaboración propia

Implementación de la Solución

El Release Manager realizó la grabación de los casos de prueba definidos, la herramienta Badboy capturó los pasos realizados en la aplicación web.

El Release Manager al finalizar la grabación, procedió a guardar el archivo con la extensión *.bx.

En una actualización del ambiente de pruebas, el Release Manager podrá hacer uso del script, el cual reproduce cada uno de los pasos guardados en el script. Ver figura 32.

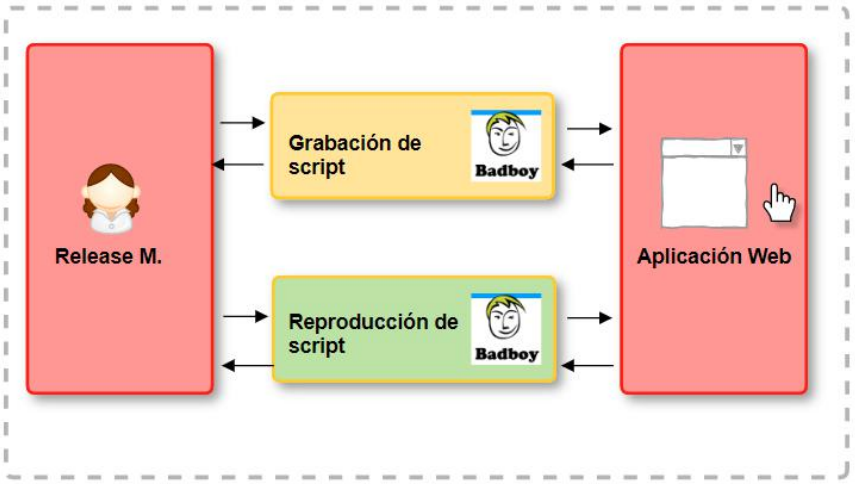

Figura 32: Diagrama de la Solución
Fuente: Elaboración Propia

Para la presente implementación se realizó la creación de la suite de pruebas basadas en el módulo de Gestión de Procesos de Pago, este módulo consta de varias secciones tal como se puede observar en la figura 33.

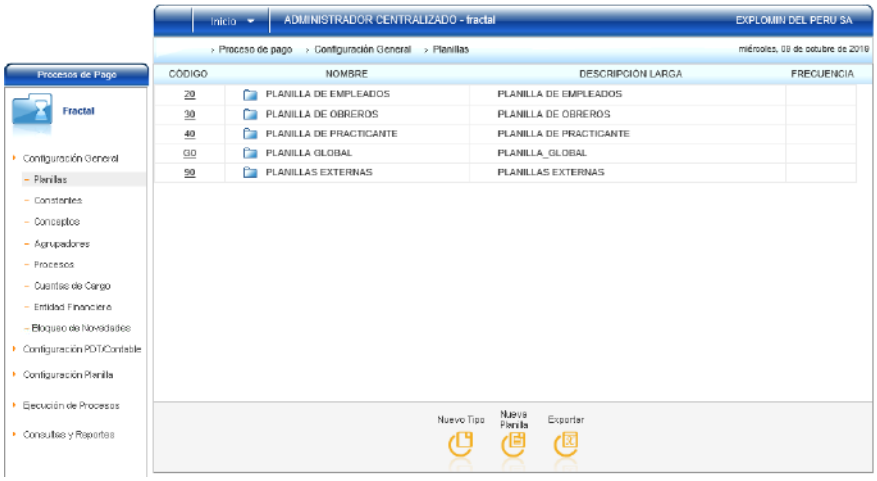

Figura 33: Modulo Proceso de Pago
Fuente: Elaboración propia

Para ello, se diseñaron los casos de pruebas que se consideron en la implementación.

Test.IniciarSesion	Código del CP	CP-001
	¿Prueba de despliegue?	Si/No
Descripción: Tipo de Prueba: Prueba de Regresión La prueba consiste en iniciar sesión con un usuario asignado para la ejecución de las pruebas Secuencia de Pasos: • Ingresar el link en el navegador IE • Ingresar el usuario • Ingresar la contraseña • Click en ingresar		
Objetivo Comprobar el inicio de sesión correcto.		
Resultados Esperados Ingresar a la aplicación		
Condición inicial Ninguno		

Figura 34: CasoPrueba.IniciarSesión
Fuente: Elaboración propia

Test.GuardarRegistroNuevo / ModificarRegistro	Código del CP	CP-002
	¿Prueba de despliegue?	Si/No
Descripción: Tipo de Prueba: Prueba de Regresión La prueba consiste en guardar un registro nuevo y/o modificar un registro Secuencia de Pasos: • Ingresar al módulo Procesos de Pago • Ingresar al fichero en la cual se creará el registro y / o se modificará • Click en guardar		
Objetivo Comprobar que el registro se haya guardado y/o modificado correctamente.		
Resultados Esperados El registro se registra y /o modifica correctamente.		
Condición inicial El usuario considerado debe tener acceso a la pestaña Procesos de Pago.		

Figura 35: CasoPrueba.RegistrarModificar
Fuente: Elaboración propia

El caso de prueba Iniciar Sesión solo se consideró al iniciar sesión al aplicativo.

El caso de prueba Registrar Nuevo /Modificar Registro se consideró para todos los ficheros del módulo Procesos de pago.

La estructura de la suite se puede ver en la figura 36 y en la figura 37 se visualiza las actividades o secuencia de pasos grabados de la sección Ejecución de Procesos.

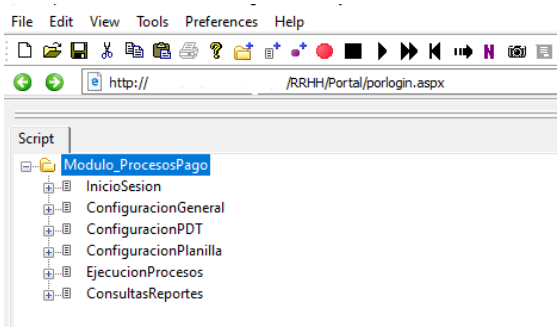

Figura 36: Estructura de la Suite
Fuente: Elaboración propia

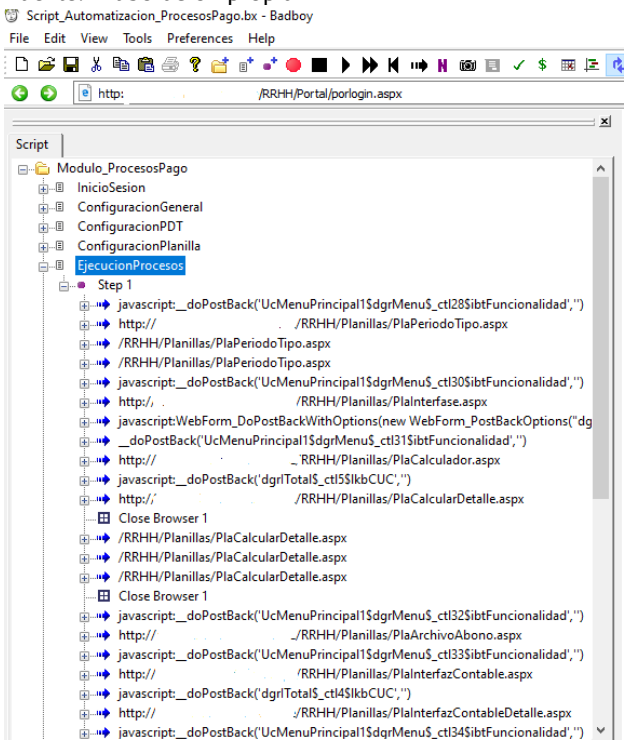

Figura 37: Actividades de un Modulo
Fuente: Elaboración propia

En la figura 38 se muestra la herramienta en ejecución, para el inicio de sesión y en la figura 39 se muestra la corrida del script para la modificación de un registro.

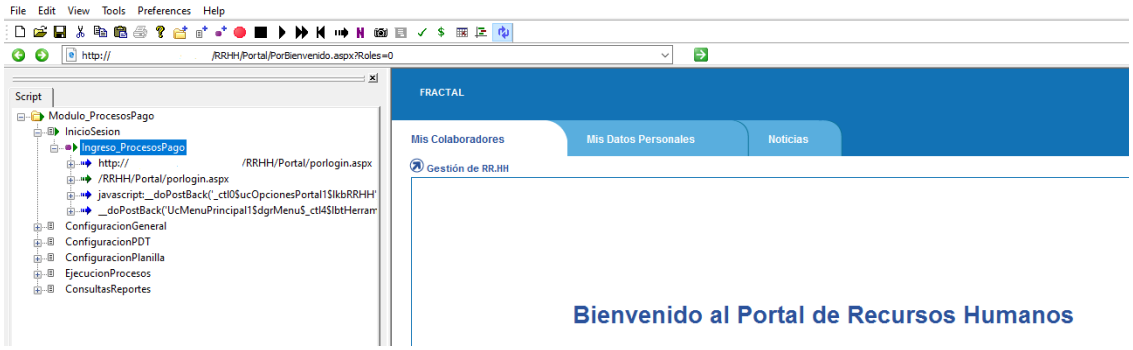

Figura 38: Herramienta en Ejecución – Inicio de sesión
Fuente: Elaboración propia

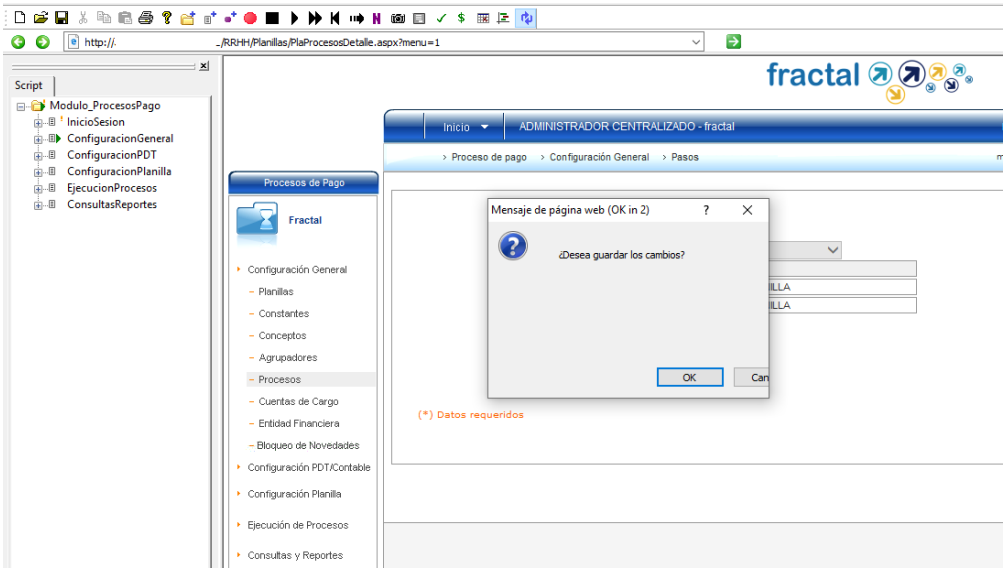

Figura 39: Herramienta en ejecución – Modificación de un registro
Fuente: Elaboración propia

Al finalizar la prueba los resultados se pueden visualizar en un reporte HTML. Ver la figura 40.

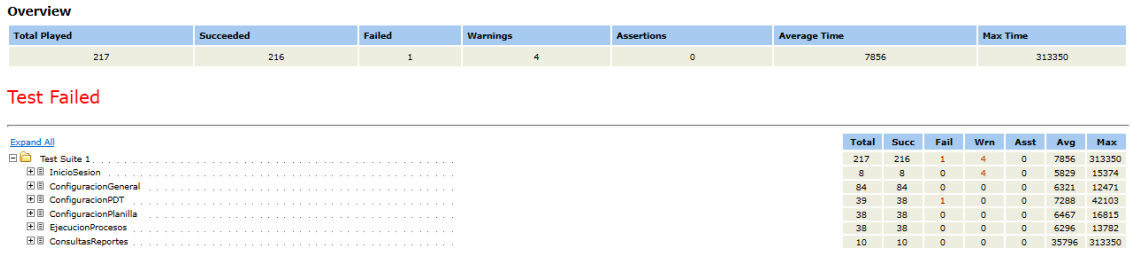

Figura 40: Reporte de Ejecución de pruebas
Fuente: Elaboración propia

Cabe indicar que antes de una actualización en Producción de un cliente se actualiza el ambiente local del cliente en mención, luego se corre el script generado en badboy el cual tiene la secuencia de pasos de registrar y/o modificar registros para todo el módulo de Gestión de Procesos de Pago luego el Release manager podrá generar el informe con la información de la test exitosos o fallidos.

Anexo 04: Evidencia de Similitud Digital

TESIS BACH GUTIERREZ KAROLY

por Karoly Gutierrez

Fecha de entrega: 30-sep-2019 08:35p.m. (UTC-0500)
Identificador de la entrega: 1183492410
Nombre del archivo: Tesis GUTIERREZ ZAPATA KAROLY GUADALUPE.docx (2.21M)
Total de palabras: 19443
Total de caracteres: 105736

TESIS BACH GUTIERREZ KAROLY

INFORME DE ORIGINALIDAD

29%	27%	1%	19%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	euskadi.eus Fuente de Internet	5%
2	Submitted to Universidad Ricardo Palma Trabajo del estudiante	3%
3	repositorio.urp.edu.pe Fuente de Internet	3%
4	Submitted to Universidad de Chile Trabajo del estudiante	2%
5	repository.udistrital.edu.co Fuente de Internet	2%
6	tesis.ucsm.edu.pe Fuente de Internet	2%
7	es.scribd.com Fuente de Internet	2%
8	documentop.com Fuente de Internet	1%
9	Submitted to Universidad Cesar Vallejo Trabajo del estudiante	1%

10	todocarrera.googlecode.com Fuente de Internet	1 %
11	Submitted to Universidad de Lima Trabajo del estudiante	1 %
12	docplayer.es Fuente de Internet	1 %
13	Submitted to INACAP Trabajo del estudiante	1 %
14	gpherrera1990.blogspot.com Fuente de Internet	1 %
15	Submitted to Universidad Continental Trabajo del estudiante	1 %
16	Submitted to Universidad Nacional de Educacion Enrique Guzman y Valle Trabajo del estudiante	<1 %
17	Submitted to Universidad Anahuac México Sur Trabajo del estudiante	<1 %
18	www.slideshare.net Fuente de Internet	<1 %
19	testingbaires.com Fuente de Internet	<1 %
20	luiscastellanos.files.wordpress.com Fuente de Internet	<1 %

21	issuu.com Fuente de Internet	<1%
22	Submitted to Universidad Estatal a Distancia Trabajo del estudiante	<1%
23	Submitted to Universidad Peruana de Las Americas Trabajo del estudiante	<1%
24	cotana.informatica.edu.bo Fuente de Internet	<1%
25	ingenieriadesoftware4u.wordpress.com Fuente de Internet	<1%
26	Submitted to Universidad Nacional de Educación a Distancia Trabajo del estudiante	<1%
27	pt.scribd.com Fuente de Internet	<1%
28	upcommons.upc.edu Fuente de Internet	<1%
29	oa.upm.es Fuente de Internet	<1%
30	www.chiletech.cl Fuente de Internet	<1%
31	eprints.ucm.es Fuente de Internet	<1%

32	Submitted to Universidad de artes, ciencias y comunicación UNIACC Trabajo del estudiante	<1%
33	repositorio.espe.edu.ec Fuente de Internet	<1%
34	revistasinvestigacion.unmsm.edu.pe Fuente de Internet	<1%

Excluir citas	Activo	Excluir coincidencias	< 20 words
Excluir bibliografía	Activo		

Anexo 05: Autorización de Publicación en Repositorio

UNIVERSIDAD PERUANA DE CIENCIAS E INFORMÁTICA
La Universidad del futuro, hoy

FORMULARIO DE AUTORIZACIÓN PARA LA PUBLICACIÓN DE TRABAJO DE INVESTIGACIÓN O TESIS EN EL REPOSITORIO INSTITUCIONAL UPCI

1.- DATOS DEL AUTOR

Apellidos y Nombres: Gutierrez Zapata, Karoly Guadalupe

DNI: 44830540 Correo electrónico: carolina.gutierrez.187@gmail.com

Domicilio:

Teléfono fijo: Teléfono celular: 949106850

2.- IDENTIFICACIÓN DEL TRABAJO Ó TESIS

Facultad/Escuela: Facultad de Ciencias e Ingeniería

Tipo: Trabajo de Investigación Bachiller ( ) Tesis (X)

Título del Trabajo de Investigación / Tesis:

Implementación de la herramienta BadBoy para la mejora de pruebas Funcionales en Aplicaciones Web.

3.- OBTENER:

Bachiller ( ) Título (X) Mg. ( ) Dr. ( ) PhD. ( )

4. AUTORIZACIÓN DE PUBLICACIÓN EN VERSIÓN ELECTRÓNICA

Por la presente declaro que el documento indicado en el ítem 2 es de mi autoría y exclusiva titularidad, ante tal razón autorizo a la Universidad Peruana Ciencias e Informática para publicar la versión electrónica en su Repositorio Institucional (http://repositorio.upci.edu.pe), según lo estipulado en el Decreto Legislativo 822, Ley sobre Derecho de Autor, Art23 y Art.33.

Autorizo la publicación de mi tesis (marque con una X):
(X) Sí, autorizo el depósito y publicación total.
( ) No, autorizo el depósito ni su publicación.

Como constancia firmo el presente documento en la ciudad de Lima, a los 23 días del mes de Noviembre de 2019.

Firma