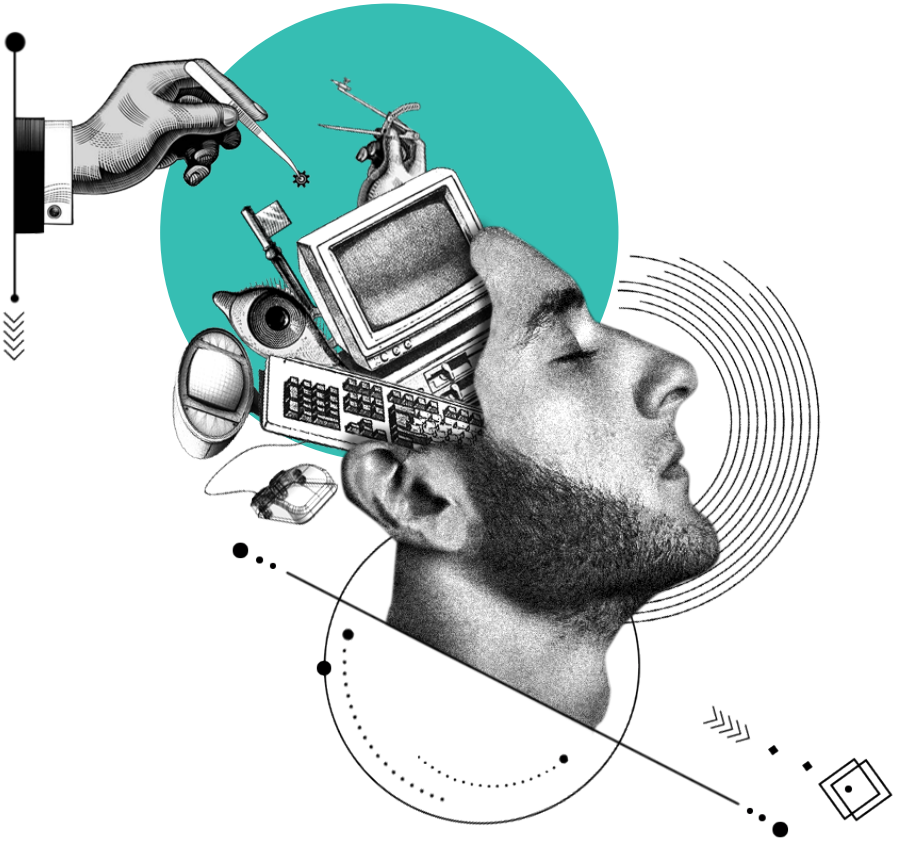


“ESTRATEGIAS PARA LAS PRUEBAS DE SOFTWARE QUE PERMITEN MEJORAR LA CALIDAD EN EL DESARROLLO DE PROYECTOS DE SOFTWARE”

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

Facultad de Ciencias de la Computación
INGENIERÍA EN TECNOLOGÍAS DE INFORMACIÓN

"Estrategias para las Pruebas de Software que Permiten Mejorar la Calidad en el Desarrollo de Proyectos de Software"

Tesis que presenta

BRENDA GUADALUPE BALDERAS TORRES

Para obtener el título de
Licenciado en Ingeniería en Tecnologías de la Información

Asesor
Dr. Abraham Sánchez López

Co-Asesor
Dra. María Beatriz Bernabé Loranca

Puebla, Puebla *Mayo 2018*

RESUMEN

Generar Software siguiendo reglas específicas es fácil, pero ¿Cómo saber si el proyecto de Software creado cumple las expectativas del usuario?, ¿Realiza tareas específicas?, ¿Se acopla perfectamente a los requisitos del cliente?, estos son algunos puntos que un equipo de trabajo debe considerar para crear e iniciar un proyecto de Software.

Existe muchos Softwares, pero un Software que se haya realizado con un proceso que aplique estrategias de calidad son muy pocos. La calidad para un Software significa que será competitivo y que será útil para los usuarios cumpliendo tareas específicas, procesos de calidad y estrategias de creación. Para lograr esto, se han generado estrategias y modelos que ayudan a diseñar procedimientos para lograr un Software exitoso.

Las pruebas de Software son un mecanismo de ayuda y un mecanismo estratégico para medir, controlar, percibir y crear la calidad de un proyecto de Software. Es imposible generar un Software libre de errores, pero si es posible tratar con esos errores y mejorarlos, por esta razón la planificación y ejecución de pruebas de Software en el desarrollo de un proyecto genera un enlace fuerte entre las estrategias que se seguirán y el momento exacto donde las pruebas de Software intervendrán en el proyecto para probarlo.

ABSTRACT

Generating Software by following specific rules is easy, but, How to know if the software project created meets the user's expectations?, Does specific tasks?, Does it fit perfectly to the customer's requirements? These are some points that a work team must consider to create and start a Software project.

There are many Software, but Softwares made with a process of quality strategies there are few. The quality of Software means competence and utility. It does specific tasks, quality processes, creation strategies. With the help of models and strategies success is achieved of Software.

Software testing is a help mechanism and a strategic mechanism which measures, controls, perceives and create the quality of a Software project. It's impossible to generate an error-free Software, but it´s possible to deal with these errors and improve them for this reason the planning and execution of Software tests in a project generates confidence.

AGRADECIMIENTOS

Las palabras son cortas para describir mi agradecimiento.

Gracias a Dios por permitir culminar otra etapa de mi vida, los caminos a veces son inciertos pero un rayo de esperanza nos permite seguir adelante.

Agradezco a mi familia quienes son mis personas favoritas que pueden existir, las personas indicadas y perfectas que son parte mí.

Agradezco a mis padres. Gracias mami por todo su esfuerzo y dedicación, por estar siempre a mi lado, por sus enseñanzas, platicas, consejos y por no dejarme sola nunca. Mami… no hay palabras para expresarte todo mi amor, y agradecimiento ¡Lo logramos! Una de las mejores cosas que tienes es brindar tu sencillez a las personas gracias Papá.

A mis abuelitos gracias por todos los momentos buenos que me han brindado por estar siempre a mi lado, por los valores, enseñanzas, risas, anécdotas…

A mis hermanos las personas con las que puedo reír de cualquier cosa, su amistad y amor una de las mejores cosas que tengo en la vida.

A mis tíos y tías hemos recorrido este camino juntos, este logro también es parte de ustedes por todo lo que me han brindado, por su amor y confianza son parte esencial de la mujer en que me he convertido. A mis primos por las risas, anécdotas, por ser cómplices, siempre hay algo que aprender de ustedes.

Agradezco a mi asesor, gracias Dr. Abraham por todas sus enseñanzas, por las pláticas, por su amistad, por enseñarnos que tenemos que ser los mejores a pesar de lo que digan, por el valor a la disciplina y por el valor a entregar los mejores trabajos, gracias por brindarme la oportunidad de pertenecer al laboratorio de MOVIS solo los mejores.

Agradezco a mi Universidad y a mi facultad me siento orgullosa de pertenecer a ustedes, es uno de mis sueños cumplidos haber pertenecido a esta institución.

Agradezco a mis amigos todos ustedes son especiales, gracias por ser parte de mi vida, por todo lo vivido y por las enseñanzas recibidas.

Agradezco a todas las personas que han sido parte de mi vida, a todas y cada una de las personas que han estado por un corto o largo tiempo, siempre hay algo que aprender. El significado de la disciplina y el amor por la música cuando algo te gusta no cuesta nada hacerlo.

“

La mejor manera de empezar
algo es dejar de hablar de ello
y empezar a hacerlo”

-Walt Disney

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

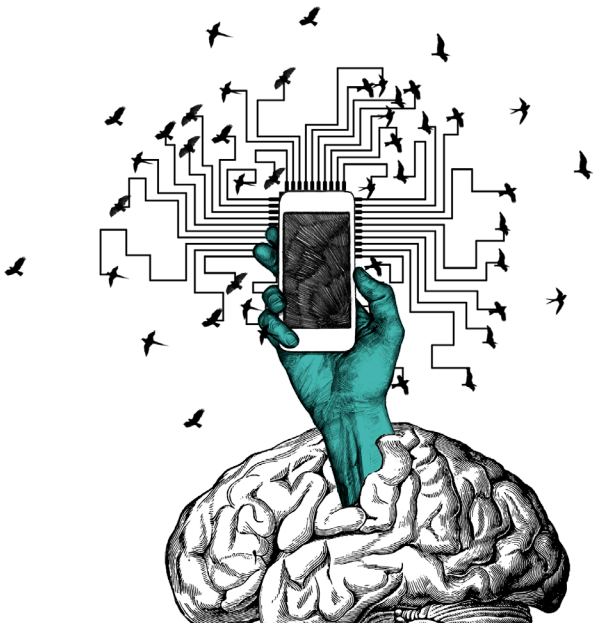
ÍNDICE DE TABLAS

ÍNDICE DE ABREVIATURAS

ABREVIATURA	DEFINICIÓN
ATDG	Automated Test Data Generation
BUAP	Benemérita Universidad Autónoma de Puebla
FCC	Facultad de Ciencias de la Computación
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Organization for Standardization
MODIHC	Human-Computer Interaction Model
PDT	Portable Data Terminal
STLC	Software Testing Life Cycle
SUT	Softeware Under Test
TDD	Test Driven Development
UML	Unified Modeling Language
V& V	Verification and Validation
XP	eXtreme Programming

CAPÍTULO 1

INTRODUCCIÓN



INTRODUCCIÓN

Para comprobar que el Software cumpla con los estándares de calidad y no existan fallas al momento de su aplicación, utilización o producción se han implementado estrategias en los procesos para proyectos que consisten en pruebas de Software.

Se realiza la tarea de obtener oportunidades para crear proyectos de Software eficaces donde son observados aspectos como el comportamiento, requisitos y los objetivos en cada iteración, tarea, fase o proceso de su realización. La creación y aplicación de pruebas de Software evalúa cada uno de los componentes de un Software con la idea de encontrar comportamientos en un Software como es el incumplimiento de requisitos, errores, resultados, funcionalidad, etc., el propósito de estas pruebas es presentar un proyecto que cumpla con la satisfacción de la interacción y los usuarios del Software.

Con el paso del tiempo la Ingeniería de Software se ha enfocado en crear proyectos que cumplan con estándares de calidad, significando así que debe cumplir con expectativas, funciones, estándares, capacidad y objetivos que el Software deberá cubrir respecto a los componentes de todo el proyecto de Software, requisitos del sistema y su funcionalidad. Para ello se han implementado métodos, estrategias y técnicas que permitan mejorar los procesos enfocados a la calidad. Realizar un Software de calidad se debe a que su diseño requiere más de la mitad de los recursos y tiempo para crearlo, con este enfoque se determina el éxito o fracaso del Software.

Las pruebas en un Software seguían una metodología donde inicialmente se generaba primero la codificación de todo el Software y posteriormente se realizaban las pruebas. La aprobación del Software se realizaba hasta terminar con la codificación de todo el Software. Con la creación de pruebas hasta el final del proyecto de Software, generalmente se manifiestan errores, procedimientos que no coinciden con el diseño, la especificación de requerimientos no se cumple, etc. Estos acontecimientos eran informados a los desarrolladores, pero, como bien sabemos cuándo se modifica un Software hay dos caminos: Se eliminan por completo los errores o se generan más. Para un proyecto de Software significa un ciclo de pérdidas porque no hay completa comunicación con el equipo de desarrolladores y el equipo de testers en todo el proceso sólo hasta el final, hay atraso en entregas, pérdidas monetarias, la calidad de Software es un enfoque que en este punto no se toma en cuenta; cuando un Software presenta todos los puntos mencionados anteriormente provoca proyectos que fracasan, provocando que el Software no se ocupe o no se entregue defraudando a los clientes y usuarios.

Con la finalidad de combatir las deficiencias en el proceso de realización de Software se generaron estrategias de prueba y métodos ágiles que, a través de su práctica pretende entregar el Software en iteraciones, módulos y funciones, como un todo garantizando que cada uno de ellos funciona a la perfección antes de lanzarlo a producción y ser utilizado.

La prueba de Software asegura que mediante el proceso de creación del Software se generan datos de entrada de los cuales se procesa información para obtener una salida de datos, aquí es donde intervienen parámetros de calidad presentando cada actividad en el proceso de evaluación de manera independiente, estos datos son observados respecto a casos de prueba, experiencias y objetivos concretos.

La ventaja de este método es que cada modificación que requiera el Software se debe realizar de manera rápida y eficaz, dicha modificación se realizará dependiendo del requisito, la demanda y necesidad del cliente con el Software; las modificaciones se realizan de manera paralela con el equipo de trabajo y evaluadores en cada fase de creación del Software. En este punto nos referimos a un proyecto de Software con calidad, entregado a tiempo, con errores detectados, funcional, resumido en una palabra “EXITOSO”.

1.1 MOTIVACIÓN DE LA TESIS

Hoy en día la oportunidad de generar y realizar proyectos realmente eficaces en los cuales intervengan parámetros de calidad, siendo que durante toda su creación sea observado, permite a los equipos de trabajo alcanzar un nivel de fiabilidad y confianza sobre el trabajo realizado para la creación de un Software.

Con base a experiencias y objetivos concretos por parte de un equipo de trabajo se realiza la tarea de observar, probar y ejecutar componentes, módulos e integraciones de un Software. Esto se hace con el fin de encontrar bugs evitando que un software falle al momento de su utilización y/o aplicación.

Las pruebas de Software se presentan en iteraciones que son concurrentes con la codificación de un proyecto de Software, aseguran que mediante diseños de casos de prueba específicos y el diseño de datos de entrada de los cuales se procesa información para obtener una salida de datos esperados, ayudarán a mantener la calidad del Proyecto de Software. Nos referimos a un proyecto exitoso y de calidad cuando a través de la observación, registro de resultados, funcionalidad y comportamiento del Software los errores son detectados a tiempo.

La utilización de estas pruebas recae al mismo tiempo en obtener un costo menor en cada modificación, por costo nos referimos al tiempo y trabajo entre las iteraciones que conforman el proyecto de Software. Las pruebas de Software centran su atención en la evaluación, depuración y un modelo de fiabilidad.

1.2 PLANTEAMIENTO DEL PROBLEMA

"El testing de programas puede ser una forma muy efectiva de mostrar la presencia de errores, pero es desesperanzadoramente inadecuado para mostrar su ausencia."

– Edsger Dijkstra

Al crear proyectos de Software siempre se pretende cubrir en su totalidad con los requisitos descritos por el cliente. Para alcanzar este objetivo importante es necesario validar cada etapa del proceso de Software utilizando descripciones de comportamiento. Detrás de esas etapas existe una arquitectura que conduce al diseño del software, esta arquitectura hace que cada etapa se analice con el fin de obtener una confianza de la funcionalidad y cumplimiento de los requisitos.

Para cumplir objetivos concretos en un Proyecto de Software se hace uso de iteraciones que son planeadas, entre cada iteración se presentan tareas específicas que se deben cumplir ya que la manera en organizar a un equipo completo que desarrollara un Software es difícil. La determinación de tiempos, tareas entre cada iteración corresponde también a un proceso de calidad, la medición de la calidad se dará a través de pruebas de Software.

Existen estrategias, modelos de calidad así mismo niveles, tipos, herramientas de implementación para pruebas de Software, en un proyecto de Software debe estar consciente de qué tipo de estrategia se debe implementar con las respectivas pruebas que al Software se deben de aplicar, posteriormente se hacen casos de pruebas donde se hacen comparaciones del comportamiento para el Software.

Sabemos que un software es percibido dependiendo del cumplimiento de cada una de las funcionalidades, entre menos fallas presente el software mayor será su utilidad y su éxito. Este enfoque incluye diferentes decisiones que deben cumplirse para alcanzar objetivos específicos.

1.3.1 GENERAL

- Proponer un marco de trabajo para la realización de pruebas de software en el desarrollo de productos de Software y con ello mejorar la calidad de estos.

1.3.1 ESPECÍFICOS

- Se proporciona un esquema donde se describen los beneficios al implementar las pruebas de Software bajo el proceso de creación del Software.
- Analizar las ventajas obtenidas en un proceso de Software a través de pruebas y como se introduce el concepto de calidad para un Software a través de estas pruebas.
- Medir la calidad del software después de la realización de las pruebas.

1.4 JUSTIFICACIÓN

Para garantizar la calidad en un Software se necesita una planificación correcta entre el tiempo, costos y recursos que se tienen para entregar un Proyecto de Software, maximizando toda su potencia y así satisfacer los requisitos con los que cuenta el Proyecto de Software.

Las pruebas de Software permiten obtener un control de todo lo que se realiza durante el proyecto de Software, observa y verifica las actividades realizadas y las compara con un resultado esperado, con un resultado real, esto se hace con el fin de encontrar anomalías en el Proyecto de Software antes de ser Integrado y entregado a sus usuarios finales.

La aplicación de estrategias a través de Pruebas de Software permite a todo un equipo de trabajo tener un amplio panorama de las funcionalidades de un Software permite la interacción entre el cliente, los desarrolladores y el equipo de pruebas.

La detección de un error a tiempo previene la pérdida de recursos, y el descontrol de un equipo de trabajo. Las pruebas de Software y estrategias de planeación proporcionan información que ayuda a administrar el curso que seguirá el Proyecto de Software. Reduce y brinda información del estado en el que se encuentra cada iteración. Las métricas son estrategias estiman la efectividad del Software.

1.5 ORGANIZACIÓN DE LA TESIS

A continuación, se presenta la organización general de este trabajo de investigación.

- *Capítulo II:* En este capítulo se muestran la ventaja de implementar pruebas ágiles en los proyectos de Software, se describen los beneficios y la manera en cómo se realiza un Software de calidad. Se describe el proceso y diseño de Software

- *Capítulo III:* Se describe el concepto de Calidad de Software, se muestran los modelos, métricas y factores de calidad en el Software, el proceso de administración de la calidad.

- *Capítulo IV:* Este capítulo describe las pruebas de Software, sus características, el plan de prueba que sigue una prueba de Software y el desarrollo de casos de prueba. Se muestran las técnicas de prueba, pruebas no funcionales y los niveles de prueba.

- *Capítulo V:* Se muestra la información de los frameworks utilizados, que son PHPUnit y Selenium, se muestra las características y su arquitectura, estos frameworks utilizan la automatización de las pruebas de Software por lo cual se describe el proceso de automatización de las pruebas de Software.

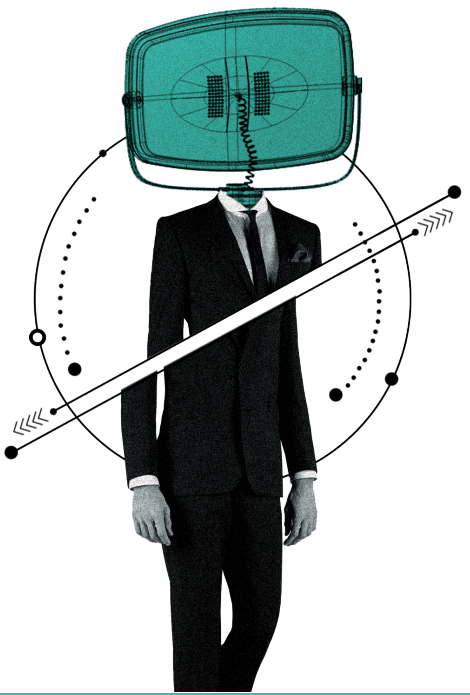
- *Capítulo VI:* El caso de prueba es imprescindible para este trabajo de investigación, se describe el contenido de la Red Social Académica, los módulos con los que cuenta y sus funcionalidades, se describe el ambiente de trabajo en el que se desarrolla, el tipo de usuarios y los diagramas que se hicieron a través de UML.

- *Capítulo VII:* En este capítulo se muestra la implementación y ejecución de las pruebas de Software, se observa el proceso de realización e implementación de pruebas en los frameworks y los resultados obtenidos en las pruebas.

- *Capítulo VIII:* Aquí se muestran las conclusiones de la investigación realizada en este trabajo de tesis, se propone un trabajo futuro con el fin de dar seguimiento a esta investigación.

CAPÍTULO 2

CALIDAD EN EL DESARROLLO DE SOFTWARE



CALIDAD EN EL DESARROLLO DE SOFTWARE

En la búsqueda constante de obtener proyectos de Software buenos y de calidad se impulsa a crear una comunicación constante con todos los involucrados en el proyecto, el cliente, los evaluadores de calidad (tester), desarrolladores etc. Se observa a detalle el comportamiento esperado del Software.

- Algunos beneficios de crear proyectos de Software que se enfatizan para obtener un alto grado de calidad en el proceso de creación son:

- Búsqueda constante de satisfacción con el cliente a través de las pruebas de Software ágiles.

- La comunicación entre los miembros del equipo produce la calidad en el código.

- Estimación del esfuerzo, evaluación, informes y tiempos en pruebas, sobre el proyecto y la calidad del Software.

- Retroalimentación continua del estado del proyecto.

- Cumplimiento de los requisitos.

En las metodologías ágiles un Sprint son iteraciones con periodos establecidos que describen tareas específicas, estas tareas son completadas para ser probadas. Las pruebas que se realizan durante todo el proceso se hacen con ciclos de desarrollo cortos, los procesos de creación del Software implican prácticas de control de calidad como parte de su desarrollo.

2.1 PROCESO DE SOFTWARE

Un proceso "es un conjunto de actividades, acciones y tareas que se ejecutan cuando va a crearse algún producto del trabajo". [16]

En la ingeniería de Software el proceso de creación de un Software se realiza a través del equipo de trabajo, determinando las actividades y tareas adecuadas que seguirán para realizar un Software, cabe mencionar que cada proceso es diferente para crear un Software.

En la creación de un Software como parte primordial, el equipo de trabajo debe entender y comprender los requisitos del sistema, una vez entendiendo lo que el cliente realmente necesita, para el equipo de trabajo será más fácil plantear una ruta de actividades para iniciar el proceso de creación del Software, el beneficio que se obtiene con el entendimiento de los objetivos son dos factores importantes:

- Obtener una base fuerte respecto a los objetivos y necesidades de satisfacción que el Software debe obtener
- El entendimiento de los requisitos permite la identificación inmediata de inconsistencias en el proceso de creación del Software.

En la *Figura 2.1* se muestra de manera general el proceso de Software.

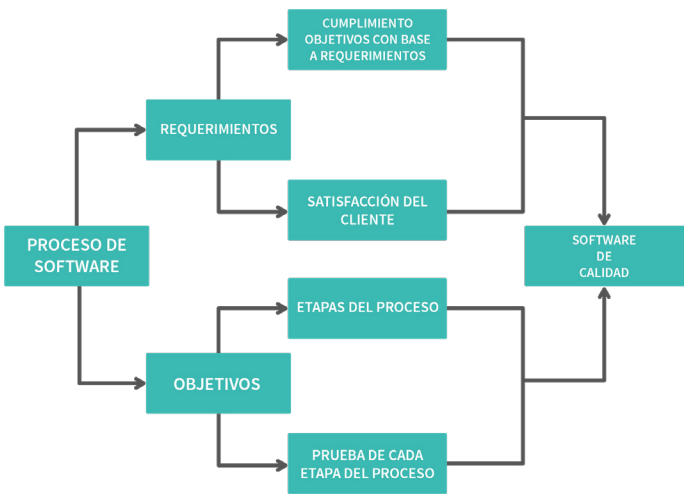


Figura 2. 1 Proceso de Software

2.2 DISEÑO DE SOFTWARE

La importancia del diseño del software se resume, en una palabra: "Calidad". [16]

Cuando no se crea un diseño previo para un Software, no se obtienen con claridad los requerimientos del Software, con el diseño se puede obtener la abstracción y comprender la funcionalidad del Software que se va a trabajar.

La calidad del Software se puede alcanzar con un buen diseño. El diseño de un Software se debe iniciar pensando en que la evolución tecnológica se mantiene constante, esto implica crear un Software adaptable, el Software debe estar abierto al cambio, las modificaciones son constantes para cubrir las necesidades en el ambiente donde sea utilizado.

Con esta evolución significativa el Software se ha vuelto una herramienta de ayuda en muchos aspectos de la vida cotidiana, por lo que el Software ya en su ambiente de producción no debe fallar porque implicaría conflictos importantes en el ambiente que el Software se desempeñe.

Cuando la utilización de un Software es imprescindible, implica que el tiempo de uso y el número de usuarios aumente. Cuando un Software es más utilizado las modificaciones de adaptación crece, los requisitos cambian, esto indica que el Software inicia un proceso de prueba para que las modificaciones hechas sean agradables a los usuarios y no presentar fallas al momento de su utilización. Lo que importa es hacer crecer el Software, evolucionarlo y mejorarlo.

2.3 PRUEBAS DE SOFTWARE ÁGILES

Las pruebas de Software Ágiles se "convierten en un componente esencial de todas las fases del desarrollo, asegurando la calidad continua del producto". *[14]* crear un proyecto de Software debe implicar una continua mejora, por tanto, los procesos de cambio deben ser dinámicos.

La presencia de pruebas ágiles en un proyecto de Software es de suma importancia, ya que las pruebas ágiles se presentan como procesos de manera iterativas e incrementales, en cada modificación que el proyecto sufre, las pruebas se van generando observando, registrando, obteniendo información a través del comportamiento del Software.

Las pruebas ágiles "combinan equipos de prueba y desarrollo en torno a los principios de colaboración, transparencia, flexibilidad y retrospección". *[15]*

La combinación entre el equipo desarrollo y el equipo que realiza las pruebas, los clientes y el usuario final, de acuerdo con los principios antes mencionados implica una retroalimentación total de un proyecto de Software, lo que provoca que todos los miembros del equipo tengan una visión general de todo el proyecto.

Las iteraciones tienen que estar probadas y deben ser consideradas como un todo esto quiere decir que cada iteración debe ser vista como el producto final, para logar la mayor calidad posible en todas las iteraciones. La comunicación es necesaria en las pruebas ágiles porque entre cada proyecto de Software surgen dudas, con ayuda de todos los puntos de vista se ameniza y se entiende lo que el Software debe de realizar.

A continuación, se muestra en la *Figura 2.2* un esquema general de cómo se realiza un proyecto de Software con pruebas ágiles.

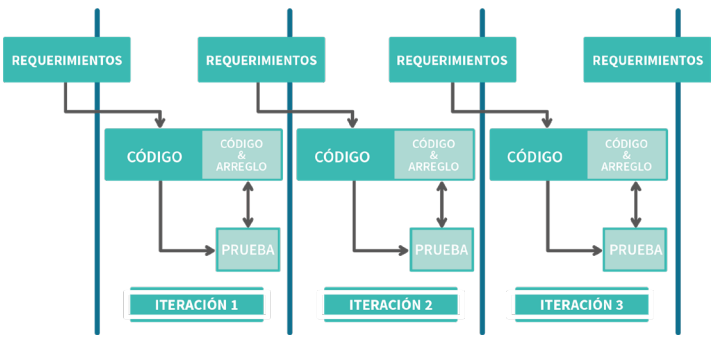


Figura 2. 2 Proceso de Pruebas de Software Ágiles [11]

2.3.1 PROCESO ÁGIL

Las prioridades y requisitos del cliente en la creación de un Software pueden cambiar en cualquier momento.

El diseño y la codificación de un Software están fuertemente enlazados, ya que al codificar un componente de Software de manera simultánea se tiene que probar en este punto que un proceso debe ser adaptable a cambios que pueden surgir en cada etapa de creación de un Software.

A continuación, se presentan 6 de los 12 principios de Agilidad: *[17]*

- La prioridad más alta es satisfacer al cliente a través de la entrega pronta y continua de software valioso.

- Los procesos ágiles dominan el cambio para provecho de la ventaja competitiva del cliente.

- Entregar con frecuencia software que funcione, de dos semanas a un par de meses, de preferencia lo más pronto que se pueda.

- El método más eficiente y eficaz para transmitir información a los integrantes de un equipo de desarrollo, y entre éstos, es la conversación cara a cara.

- La medida principal de avance es el software que funciona.

- Las mejores arquitecturas, requerimientos y diseños surgen de los equipos con organización propia.

Como se observa en estos principios se hace énfasis en la entrega de un proyecto de Software de calidad que cubra todas las necesidades del cliente así como las actividades y el comportamiento que un trabajo en equipo debe cumplir.

2.4 DESARROLLO BASADO EN PRUEBAS

El desarrollo basado en pruebas (TDD) por sus siglas en Inglés también llamado diseño basado en pruebas es un enfoque donde el equipo de trabajo debe ser muy dinámico, como su nombre lo indica Ágil. Existe la comunicación directa entre el equipo de trabajo, el equipo de evaluadores (testers), los clientes y usuarios con el fin de obtener una mejora continua. Para crear este tipo de desarrollo se crean iteraciones con respectivas tareas además se limita el objetivo general del Software, las planeaciones y el equipo de trabajo realiza este tipo de desarrollo con el fin de crear y fijar un alcance entre cada iteración así será más fácil lograr el alcance de todo el Software si éste se muestra por partes y después se realiza su Integración.

Test Driven Development (TDD) es una práctica iterativa de diseño de software orientado a objetos, que fue presentada por Kent Beck y Ward Cunningham como parte de XP *[13]*

Kent Beck menciona algunos de los beneficios de utilizar el Desarrollo Basado en Pruebas *[12]*

- La calidad del Software aumenta
- El código es reutilizable
- Multiplicación entre los miembros del equipo

2.4.1 CICLO TDD

- El ciclo se inicia con la elección de un requisito específico del Software.

- Posteriormente se crea el diseño de una prueba respecto al requisito que se eligió.

- Se inicializa la prueba, si la prueba no falla el requisito ha sido entendido y previamente codificado, el Software presenta una funcionalidad correcta.

- Cuando se manifiesta algo fuera del Requisito o el Software falla, se implementa una prueba por el equipo de evaluadores (testers), se crean los datos de entrada para dicha prueba y se implementa la automatización. La automatización de las pruebas de Software hace la comparación entre los resultados obtenidos y los resultados esperados.

- La refactorización permite depurar Software repetido, las pruebas que pueden ser ocupadas se almacenan en iteraciones posteriores para no realizar doble trabajo.

En la *Figura 2.3* se presenta el ciclo de las actividades que se realizan dentro de un proyecto de Software con TDD.

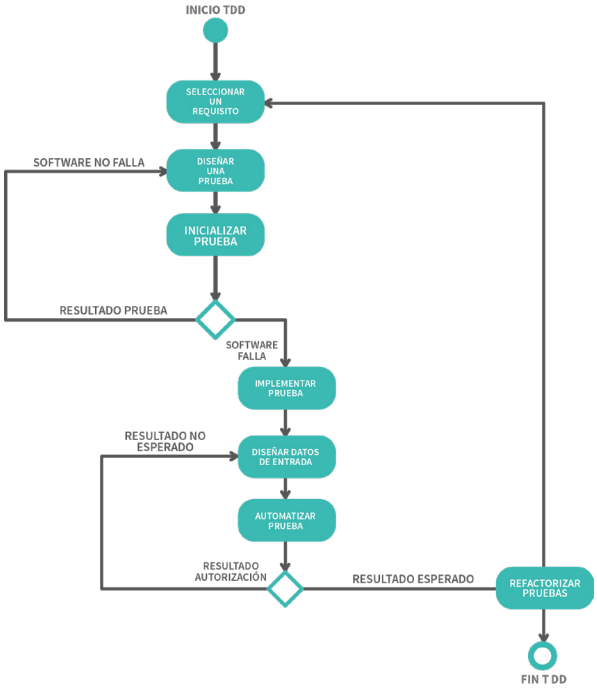


Figura 2. 3 Ciclo TDD

2.5 VERIFICACIÓN Y VALIDACIÓN

Entre 1983 y 1984 surgen los conceptos verificación y validación donde se implementan métodos para desarrollar un Software, estas implementaciones de métodos se enfocan a verificar la calidad del Software.

La verificación y validación del software son actividades de aseguramiento de la calidad del software que apuntan a asegurarse de que el sistema de software se desarrolle de acuerdo con un proceso de desarrollo y satisfaga las necesidades del cliente. *[8]*

Figura 2. 4 Flujo V & V

En la *Figura 2.4* Las actividades del proceso V & V se inician con las actividades de verificación en el proceso de Software y posteriormente se realiza el de validación.

Como se observa la verificación y validación de un Software es un proceso importante, donde se realizan evaluaciones, pruebas de Software, análisis de requisitos y se observa la calidad del Software.

A continuación, se mencionan algunas técnicas de V & V dentro de un proyecto de Software *[7]*

Informales: Se basan en el razonamiento humano.

- Pruebas de escritorio
- Tutoriales

Estáticas: Evalúa principalmente el código fuente del Proyecto de Software.

- Depuración
- Transición de Estados
- Análisis de Sintaxis

Dinámicas: Evalúa a través de pruebas se observa el comportamiento del Software.

- Pruebas de Aceptación
- Pruebas de Seguridad
- Pruebas Funcionales (Black - Box)
- Pruebas de Regresión
- Pruebas Estructurales (White - Box)

Formales: Se basan en pruebas matemáticas.

- Afirmación Inductiva, se afirma que la conclusión es probable pero no es absolutamente verdadera

2.5.1 VERIFICACIÓN

IEEE determina la verificación como *"El proceso de determinar si los productos de una fase determinada de un proceso de desarrollo de software cumplen o no los requisitos establecidos durante la fase previa [IEEE83a]" [9]*

Se hace referencia a la calidad del proyecto de Software, la validación representa en este caso la calidad y el valor agregado que el usuario le dará al Software, representa si lo creado fue correctamente hecho. A través de la verificación se evalúan los requisitos del proyecto, se hacen revisiones y su enfoque es la calidad que se puede obtener con este tipo de evaluaciones.

2.5.2 VALIDACIÓN

IEEE define la validación como: *"El proceso de evaluación de software al final de su proceso de desarrollo para garantizar el cumplimiento de sus requisitos [IEEE83a]". [9]*

Se hace referencia y una comparación respecto a las funciones y el porcentaje de cumplimiento respecto a los requerimientos y especificaciones del proyecto de Software. Se evalúa la funcionalidad del proyecto creado en sus diversas fases esto se realiza a través del equipo de evaluadores (testers) creando pruebas de Software.

2.6 V & V EN EL PROCESO DE CREACIÓN DE UN SOFTWARE

En el proceso de Software se observa cómo la validación y verificación intervienen en cada fase de creación de un proyecto de Software y los beneficios que con este proceso se puede obtener al implementar esta estrategia.

2.6.1 REQUISITOS

En este punto implementar la estrategia V & V en los requisitos se da a la tarea de revisar todas las especificaciones del cliente, estas representan los requisitos del Software. Ayuda al equipo de trabajo a entender completamente la funcionalidad del Software y cuál será el objetivo o necesidades que debe cumplir o ejecutar. Con el Análisis de requisitos se pretende erradicar la inconsistencia entre requisitos, la ambigüedad y redundancia. A continuación, se muestra el proceso de Requisitos de Software *Figura 2.5.*

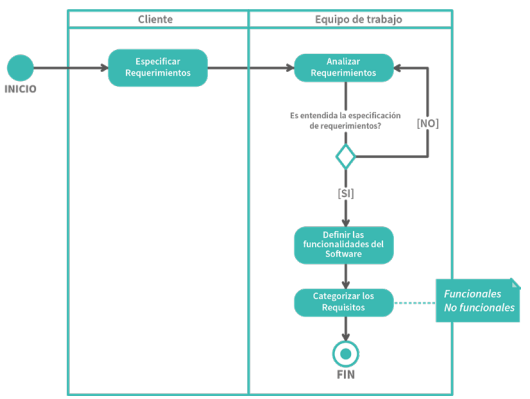


Figura 2. 5 Requisitos del Software

2.6.2 DISEÑO

La verificación de alto nivel se asegura de que cada requerimiento se realiza por algunos módulos en el diseño y cada módulo en el diseño es necesario para satisfaciendo algunos requisitos. Con el diseño de la aplicación a través del modelado UML del Proyecto de Software se hace una representación en diferentes niveles de abstracción. En la *Figura 2.6* se observa el proceso de Diseño.

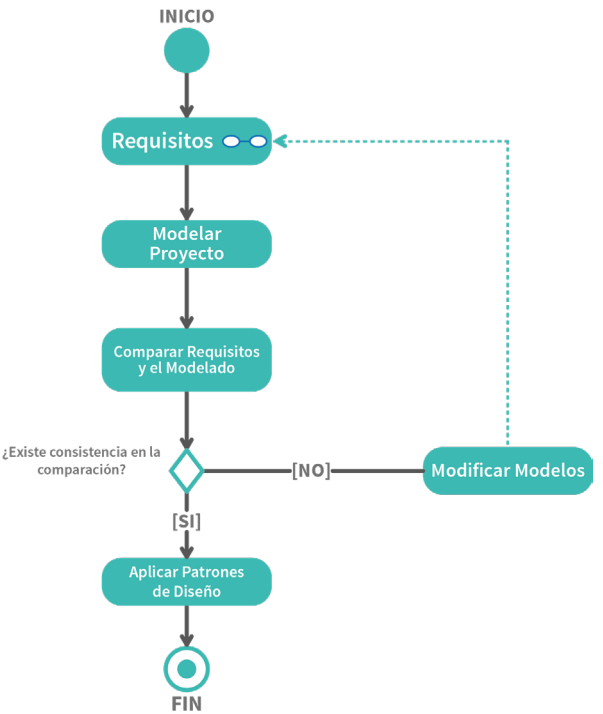


Figura 2. 6 Proceso de diseño del Software

2.6.3 IMPLEMENTACIÓN

Mediante la Implementación en el proceso de creación de un Software se hace la codificación de la aplicación, esta codificación dependerá del estándar que se implemente por parte del equipo de trabajo, se hace una comparación ahora con el modelado de la aplicación y la implementación del diseño codificado en módulos del Software. Se detecta la función correcta de la estructura de datos, el uso correcto de los operadores lógicos, tiempo de respuesta y la calidad respecto a la sintaxis. En la *Figura 2.7* se muestra las actividades de la fase de Implementación respecto al proceso V & V.

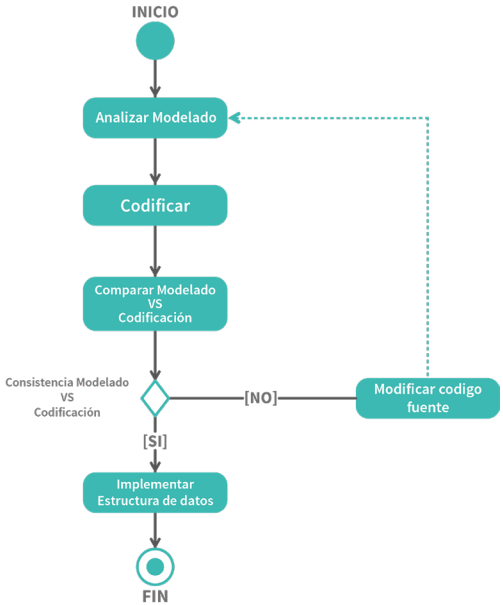


Figura 2. 7 Implementación V & V

2.6.4 INTEGRACIÓN

Es la fase final para el proceso V & V como su nombre lo indica es la integración total de todos los módulos que conforman el Software; se integran las interfaces a través de parámetros observando la secuencia entre los módulos, además de las tareas específicas que con datos de entrada se interactuar de una interfaz a otra.

CAPÍTULO 3

CALIDAD DE SOFTWARE

CALIDAD DE SOFTWARE

Hablar de calidad en un Software hace referencia a cumplir las especificaciones, necesidades y tareas específicas del cliente, el concepto de calidad en este enfoque es abstracto porque se determina a través del valor adquirido por el usuario al utilizar un Software.

Un Software debe cumplir y realizar tareas específicas con el fin de brindar eficiencia y confiabilidad al usuario. La calidad del Software se construye desde el punto en el que un Software es redactado a través de las especificaciones del cliente. Las especificaciones del cliente determinan la estructura y planificación de las actividades para lograr que un Software sea codificado y probado de manera óptima.

Cuando un Software es planificado, creado y codificado con estándares y perspectivas de calidad se disminuye el riesgo de que un Software no sea funcional o que el usuario se encuentre con aspectos del Software que estén fuera de sus expectativas provocando el rechazo inmediato de utilización del Software. Para el equipo de desarrollo y el proceso de prueba de Software, estos factores de riesgo tienen como consecuencia realizar modificaciones cuando un Software ya está en uso, incluso cabe la posibilidad de que los cambios efectuados no sean realmente los que se esperan provocando que otros errores aparezcan.

Los enfoques con los que la calidad del Software será medida de manera objetiva se buscan la perfección absoluta con el fin de crear un Software que pueda ser utilizado y aceptado por los clientes, la conclusión de esta aceptación significará que ha valido la pena el costo del Software porque se obtuvo un beneficio, y para el equipo de trabajo significará que el trabajo realizado, los planes y creación de tareas fueron optimas y correctas.

IEEE define la calidad de Software como:

"La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario." [2]

La calidad de Software es un efecto dominó, esto se debe a que, si las planificaciones de tareas respecto a los requisitos del cliente no son entendidas, la codificación será errónea porque el equipo de trabajo no entiende lo que el cliente necesita y por tanto el proceso de pruebas de Software no aplican en ninguna etapa de creación porque no se sabe que es lo que realmente se tiene que probar.

3.1 CALIDAD DE SOFTWARE

Cuando un equipo de trabajo está enfocado en brindar un Software de calidad a sus clientes o usuarios debe enfocarse en detectar deficiencias y en explicar por qué un Software presenta fallas y dónde es que se encuentran estas fallas. Cualquier Software debe ser capaz de ser modificado estableciendo criterios de pruebas para observar si un Software cumple objetivos establecidos.

La calidad de Software se define como: "la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente". *[16]*

Entender las necesidades del cliente y principalmente del usuario no es una tarea fácil, la perspectiva de un Software de calidad es prever que un Software tienda a fallar y si el caso de falla se presenta el implementar métricas para medir la calidad ayudará a encontrar errores a tiempo. En el proceso de creación de cualquier Software se debe tener una buena planificación, control en los procesos y aseguramiento en los procesos con el fin de encontrar y mejorar fallas a tiempo.

Como se observa el cumplimiento de los objetivos de Software, la satisfacción con el cliente inspecciones y procesos de creación del Software son indicativos de calidad.

3.2 ADMINISTRACIÓN DE LA CALIDAD DE SOFTWARE

ISO 9000:2000, considera que la gestión de calidad consiste en la realización de actividades coordinadas que permiten dirigir y controlar una organización en lo relativo a la calidad.

En la búsqueda de calidad para un Software se observa el proceso de creación, intervienen actividades y tareas específicas que un grupo de trabajo deberá realizar. Estas actividades y tareas buscan el cumplimiento de un objetivo específico, para obtener una mejor visión de la entrega total del Software.

La gestión de la calidad asegura que un Software cumplirá con los objetivos descritos en la especificación de requisitos y las expectativas del cliente. Cuando un Software es entregado y se pone en marcha su funcionamiento, tanto el cliente como el usuario final determinaran la calidad de Software.

La gestión de la calidad realiza mediciones donde se observa el grado de satisfacción del cliente, se sigue respecto a los requisitos y planeaciones del Software y el grado con el que se ha conseguido.

3.3 MÉTRICAS DE CALIDAD DE SOFTWARE

Son herramientas para que cualquier persona involucrada en la ingeniería de software entienda diversos aspectos de la base de código y el progreso del proyecto. *[23]*

Las métricas se enfocan a la calidad del Software y del proceso con el cual se llevará a cabo el proyecto de Software, entre las observaciones que realizan las métricas son el rendimiento y estimación de costos. Las métricas son propias de cada modelo de calidad y se crean para medir los criterios definidos en él.

Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado. *[21]*

A través de las métricas y los modelos de calidad de Software se evalúa la efectividad y mejora de los procesos, las características del producto de Software y el total del esfuerzo. Existen tres tipos de métricas que son: del producto, proceso y proyecto. Uno de los principales enfoques de la implementación de métricas es observar líneas de código que pueden ser reutilizables y con ello se modifican las estimaciones y mejora el diseño de un Software.

- ***Métricas del producto:*** muestran las características del Software.
- ***Métricas del proyecto:*** muestran el estado y las ejecuciones de las tareas durante la creación de un Software.
- ***Métricas del proceso:*** muestran las características y resultados en cada proceso de Software.
- ***Meticas de complejidad:*** determinan el volumen y tamaño de un Software.
- ***Métricas de calidad:*** miden la exactitud, pruebas y mantenimiento de un Software.

Como se observa una métrica mide la calidad de los atributos o entidades de un Software, algunos de estos atributos son:

- ***LOC:*** Líneas de Código
- ***HPD:*** Horas de Programador

3.4 MODELOS DE CALIDAD DE SOFTWARE

Los Modelos de Calidad son aquellos documentos que integran la mayor parte de las mejores prácticas, proponen temas de administración en los que cada organización debe hacer énfasis, integran diferentes prácticas dirigidas a los procesos clave y permiten medir los avances en calidad. *[18]*

Existen modelos que describen la calidad de Software y su descomposición, con el fin de poder mediar la calidad respecto a un proyecto de Software, a través de los modelos se puede determinar el esfuerzo de mantenimiento. Los modelos proporcionan una medida de confiabilidad para predecir la fiabilidad en el Software. Los modelos de calidad proporcionan información intentando predecir la confiabilidad de los datos, la observación del proyecto de Software también ayuda a determinar y planificar cuantas pruebas se aplicarán.

Un modelo de calidad se implementa con el objetivo de describir, evaluar y/o predecir la calidad. Independientemente de la técnica con que el proyecto ha sido modelado. *[19]*

Los modelos de calidad de Software tienen una estructura jerárquica, en los modelos de calidad se observa los factores de calidad. Los factores son evaluados desde el punto de vista del usuario, posteriormente se muestran los criterios de calidad aquí, se evalúa el producto de Software y por último se obtienen las métricas. Con las métricas se mide la calidad con la que el Software será presentado.

3.4.1 MODELO MCCALL

Este modelo fue presentado en 1977, se basa en 11 factores de calidad presentado por Jim McCall, desarrollado para la Fuerza Aérea de los EE. UU *[24]*

En la *Figura 3.1* se observa el árbol de factores de McCall

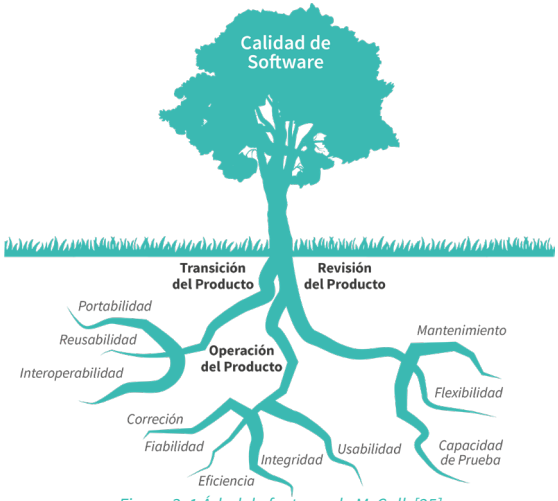


Figura 3. 1 Árbol de factores de McCall [25]

Los factores representan las características del sistema, el modelo McCall lo divide en tres perspectivas:

- ***Transición del Producto:*** Capacidad del Software para adaptarse a nuevos escenarios, la comunicación con otros sistemas y la oportunidad de ser utilizado por partes.

- ***Operación del Producto:*** Observación de las funcionalidades que contiene el Software. Se observa si el Software hace lo que se quiere, la seguridad y su ejecución.

- ***Revisión del Producto:*** Capacidad que tiene el Software de ser cambiado. Intervienen funciones como las pruebas, modificaciones y corrección de errores.

Este modelo contiene tres perspectivas para identificar la calidad de un Software, que son las operaciones del producto, revisión del producto y su transición. El Software es sometido a pruebas para observar si es capaz de someterse a cambios y puede ser adaptado a diferentes entornos.

Como se observa estas características contienen factores, estos factores son abstractos por lo que cada uno contiene criterios de calidad.

Los criterios de calidad son un atributo de una calidad respecto al factor que está relacionado con la producción de software y su diseño. [25] Estos criterios de calidad recaen en una métrica de calidad que son atributos que son medibles.

3.4.1.1 FACTORES DE CALIDAD DE MCCALL

- ***Mantenimiento:*** Mide la capacidad del esfuerzo para corregir errores, las buenas prácticas se hacen presentes como un buen diseño y una metodología.
- ***Flexibilidad:*** Capacidad de realizar y adaptación a cambios que se realizan al Software.
- ***Capacidad de Prueba:*** Planificación e implementación de las pruebas, involucra procedimientos, tiempo, errores.
- ***Portabilidad:*** Nivel de adaptación a nuevos escenarios de implementación.
- ***Reusabilidad:*** Adaptación del Software o componentes del Software a nuevos contextos.
- ***Interoperabilidad:*** Adaptación con otro tipo de Softwares o aplicaciones de Softwares externos.
- ***Corrección:*** Grado en el que el Software cumple con la especificación de requerimientos.
- ***Fiabilidad:*** Recuperación del Software a situaciones no esperadas.
- ***Eficiencia:*** Uso específico y correcto de recursos.
- ***Integridad:*** Seguridad del Software respecto a datos y su utilización.
- ***Usabilidad:*** Facilidad de aprendizaje para manejar el Software.

En la *Figura 3.2* se muestra las perspectivas, factores y criterios de calidad del modelo McCall.

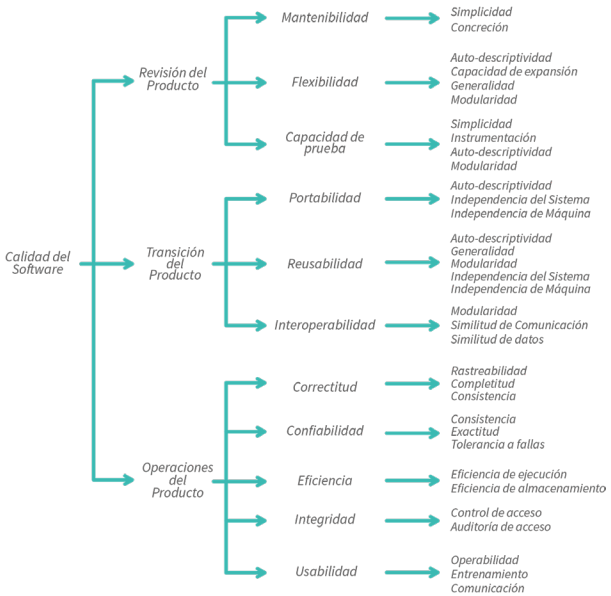


Figura 3. 2 Modelo McCall [3]

3.4.2 MODELO BOEHM

Presentado por Barry W. Boehm, este modelo evalúa cuantitativamente la calidad de Software. *[24]*

Este modelo es jerárquico y se enfoca a tres tipos de niveles: Características de alto nivel, características de nivel medio y características primitivas.

El alto nivel muestra las características del Software con base a cómo lo percibe el usuario, se observa la facilidad de uso, su confiabilidad y eficiencia. El mantenimiento se hace presente para observar si el Software puede ser usado como es presentado y qué tan confiable es en adaptarse a nuevos entornos.

En el segundo nivel se puede observar los factores de calidad, estos factores son descritos posteriormente.

En las características primitivas, que representan el nivel más bajo, Boehm determinan las características del Software que determina la definición de las métricas.

3.4.2.1 FACTORES DE CALIDAD DE BOEHM

A continuación, se describen los factores de calidad del Modelo de Boehm.

- ***Portabilidad:*** Este factor se enfatiza al código y corresponde a la facilidad en que puede ser operado.

- ***Fiabilidad:*** Se enfoca a observar las funcionalidades del Software.

- ***Eficiencia:*** El código hace uso de los recursos necesarios para ser operado.

- ***Usabilidad:*** Se enfatiza la observación de la interfaz del Software con ello la capacidad y facilidad de uso es observada.

- ***Testabilidad:*** El Software debe de cumplir con los requisitos planteados para su creación.

- ***Comprensibilidad:*** Cuando se crea un Software uno de los principales objetivos del diseño es que el usuario comprenda la estructura y el propósito del Software.

- ***Flexibilidad:*** Este es un factor del Software capaz de transformarse o cambiar para cubrir otros requisitos que el cliente solicite.

En la *Figura 3.3* se muestra la estructura del modelo de calidad de Boehm

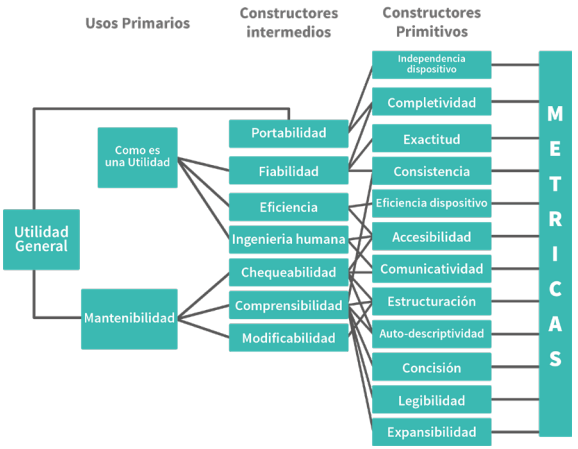


Figura 3. 3 Modelo Boehm [4]

3.4.3 ISO/IEC 9126

Creado en 1991 el nombre formal es ISO 9126 Software Product Evaluation: Quality Characteristics and Guidelines *[27]* Este modelo se realizó con el fin de evaluar productos de Software, se definen seis características de calidad. En base a este estándar se especifican los requisitos funcionales y no funcionales del Software. Se identifican los requisitos del Software para detectar los objetivos tanto del diseño como de las pruebas de Software.

El estándar describe la calidad del proceso donde se refleja la calidad del Software y la calidad del Software se refleja en el uso del Software, este es un ciclo de retroalimentación.

Este estándar describe la calidad de Software en dos partes:

- ***Calidad interna y externa:*** Aquí se especifican seis factores de calidad.

- ***Calidad en uso:*** Representa los criterios de calidad, específicamente son cuatro por cada factor.

3.4.3.1 FACTORES DE CALIDAD DEL ESTÁNDAR 9126

A continuación, se describen los factores de calidad para el estándar ISO 9126:

- ***Funcionalidad:*** Se observan las funciones satisfaciendo los requisitos del Software, se observa la adecuación que representa las tareas específicas que debe realizar el Software, los resultados, la interacción con otros sistemas y la accesibilidad de los datos.
- ***Confiabilidad:*** Se determinan condiciones establecidas, son observados, el desempeño, tiempo y la tolerancia a fallos.
- ***Eficiencia:*** Estos identifican los requisitos necesarios, se observan los tiempos de respuesta en conjunto con el procesamiento y el rendimiento.
- ***Usabilidad:*** Se enfatiza la obtención de los atributos necesarios para utilizar el Software, se observa el reconocimiento de los elementos del Software para comprenderlos y utilizarlos. El Software debe ser atractivo en cuanto a su interfaz se refiere.
- ***Mantenibilidad:*** Con este factor se hacer énfasis en la corrección de errores oportunos y modificaciones que el Software pueda sufrir, inclusive determinar la causa de la falla
- ***Portabilidad:*** Adaptación del Software por ser adaptado a diferentes plataformas.

ISO 9126 se reconoce cuatro factores de calidad de uso, estos factores se reflejan mediante la aceptación del Software por parte de los usuarios.

- ***Eficacia:*** Realización de tareas específicas dentro del Software.
- ***Productividad:*** Se emplean los recursos en un contexto específico.
- ***Seguridad:*** Utilización de datos específicos de los usuarios y su integridad.
- ***Satisfacción:*** Satisfacción por parte del usuario en la utilización del Software.

En la *Figura 3.4* se muestra la estructura de estándar ISO/ IEC 9126

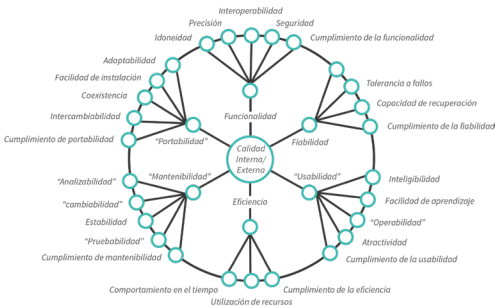


Figura 3. 4 ISO/IEC 9126 [28]

3.4.4 MODELO FURPS

Presentado por Robert Grady en 1987 *[26]* el modelo se realizó respecto a la clasificación de requisitos funcionales y no funcionales.

Este modelo se realizó con la finalidad de reducir los riesgos en cada fase de un Software, se estructuraron cinco factores de calidad que a continuación se describen. Este modelo se creó con el fin de demostrar que estos factores eran las características que un software debe cumplir

Los factores de calidad que representan este modelo son entendibles, este modelo toma en cuenta al producto y al proceso con el que fue realizado.

3.4.4.1 FACTORES DE CALIDAD DEL MODELO FURPS

Estos son los cinco factores que representan al modelo FURPS, como mencionamos anteriormente este modelo hace referencia a los requisitos, la manera en cómo los factores se dividen respecto a los requisitos es la siguiente.

- *Requisitos funcionales*
  - Funcionalidad
- *Requisitos no Funcionales*
  - Usabilidad
  - Confiabilidad
  - Rendimiento
  - Soporte

FURPS define sus factores de la siguiente manera:

- *Funcionality (Funcionalidad):* Implica la seguridad en el Software.
- *Usability (Usabilidad):* Incluye factores humanos y la coherencia entre transacciones respecto a la interfaz.
- *Reliability (Confiabilidad):* Capacidad de recuperación a fallos
- *Performance (Rendimiento):* Con este factor de calidad se obtiene resultados de la velocidad, eficiencia y consumo en el Software.
- *Supportability (Soporte):* Se especifica la adaptabilidad, la configuración y compatibilidad con otros sistemas.

En la *Figura 3.5* se muestra la estructura del modelo F.U.R.P.S.

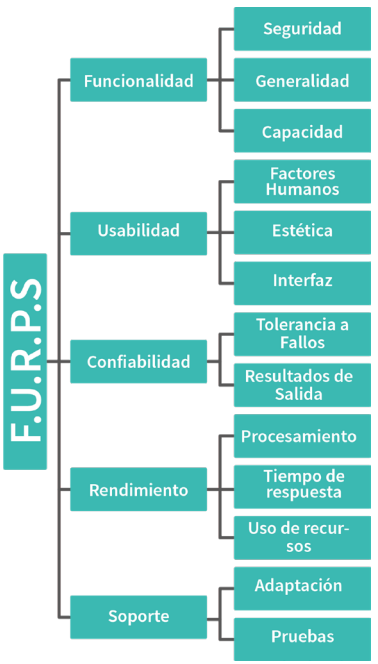


Figura 3. 5 Modelo FURBS

3.4.5 MODELO V

"En la práctica un Modelo en V puede tener diferentes niveles de desarrollo en función del proyecto y el producto software" [31]

Este modelo está definido en cada etapa del proceso de desarrollo del Software. Cada nivel de prueba de Software se aplica a cada proceso o actividad en el desarrollo de Software.

- ***Prueba de Aceptación:*** Estas pruebas se aplican en el proceso de la definición de Requerimientos. Estas pruebas se producen mediante la evaluación del Software y el cumplimiento de requerimientos.

- ***Prueba del Sistema:*** es la prueba del Software previamente desarrollado, analizando su uso y comportamiento, detectando defectos generales del Sistema.

- ***Prueba de Integración:*** Permite la verificación y observación del comportamiento entre módulos del Software, se emplean en el diseño preliminar del Software.

- ***Prueba de Unidad:*** Se realiza la evaluación de cada uno de los Componentes o módulos del sistema, dicha prueba se presenta en el proceso de diseño detallado para el Software.

En la *Figura 3.6* se muestra el diagrama que conforma el modelo V.

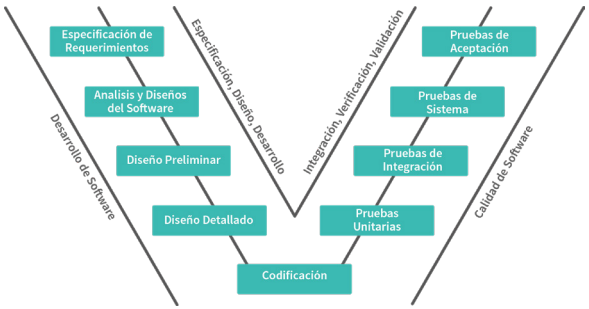
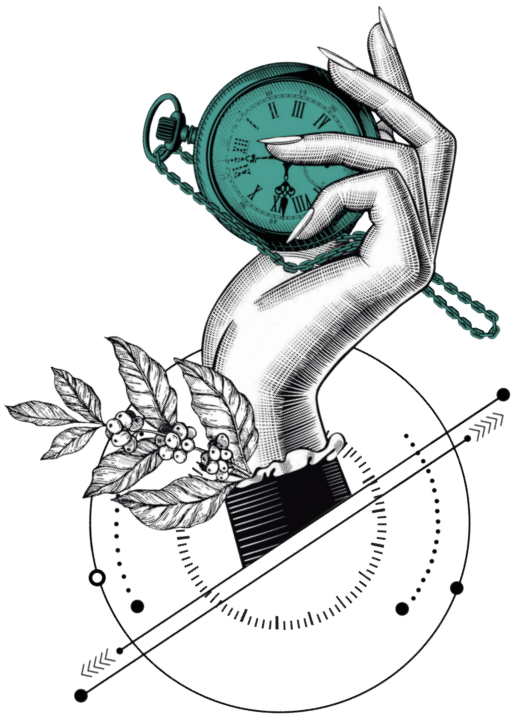


Figura 3. 6 Modelo V

Este modelo es una extensión del modelo en cascada, se observa que la parte izquierda representa las fases del proyecto respecto a su desarrollo y la parte derecha corresponden a la aplicación de pruebas a realizar.

CAPÍTULO 4

PRUEBAS DE SOFTWARE



PRUEBAS DE SOFTWARE

Uno de los motivos por el cual se realizan pruebas es para obtener un grado de calidad en la creación de un Software a través de la búsqueda de errores que se pueden generar en cada proceso de creación del proyecto. La calidad se centra en cumplir con las expectativas del cliente.

Las pruebas para demostrar la calidad de un Software se realizan durante el proceso de creación antes de entregar un Software al cliente o usuario final. A través de las pruebas se observa el comportamiento del Software, esto se hace con entrada y salida de datos específicos. La ventaja de estas pruebas es encontrar fallas, inconsistencias o errores que provoquen que un Software sea señalado como un Software que no sirve, un Software que no cumple con las expectativas de sus clientes es un Software considerado no exitoso.

Las pruebas de Software son una constante secuencia de cambios en el desarrollo de un Software, cada secuencia de pruebas contiene una estrategia donde se encuentra un plan específico para los casos de prueba. Estos casos de prueba serán ejecutados para obtener y observar el comportamiento del Software, el seguimiento de pruebas verifica el cumplimiento de los requisitos y objetivos del Software, es aquí donde la calidad del Software se hace presente.

Nos remontamos a las primeras pruebas de Software donde estaban orientadas a la corrección única y exclusivamente del código, en este proceso sólo se aseguraba que un Software funcionara de una manera correcta, por otro lado, surge el concepto de "testing" donde se inicia con la verificación y aseguramiento del Software donde se debían realizar tareas para resolver un problema o una tarea específica dentro del Software.

Los años entre 1979 y 1982 *[1]* el testing sufre una modificación radical, ya que se hace referencia a que el testing se debía ejecutar con la intención de encontrar errores para mejorar un Software. Para la búsqueda de fallas en el Software surge el concepto de Caso de Prueba, eran pequeñas pruebas para verificar una tarea específica en el Software.

En la creación de pruebas de Software se prevé la mejora en el desarrollo del Software, la evaluación, la productividad del equipo de trabajo, el mantenimiento de un software disminuye, prevé la funcionalidad de un Software, cumple con las especificaciones. En la creación de un proyecto de Software se debe impresionar al cliente entregando un buen Software que valga la pena su creación y utilización. Cuando los puntos anteriores no se cumplen, todo el esfuerzo, el trabajo y dedicación del equipo de trabajo, las estrategias implementadas, la metodología, el diseño, y la aplicación de una Ingeniería de Software, habrá sido en vano.

4.1 PRUEBA DE SOFTWARE

Hoy en día las pruebas de Software son sometidas a retos de adaptación, esto se debe a los diferentes tipos de plataformas en donde un Software se emplea. Las pruebas de Software representan un medio que conduce a la calidad de cualquier proyecto de Software, a través de las pruebas de Software se prevé combatir problemas como sobrecostos en la creación de un Software, la falta de organización y estimación de las actividades realizadas entre sus procesos.

El estándar IEEE 610.12-1990 define el termino de prueba como: "Actividad en la que un sistema o componente se ejecuta bajo condiciones específicas, los resultados se observan y registran, se realiza una evaluación de algún aspecto del sistema o componente" *[2]*

Esta definición hace referencia a que mediante las pruebas se pueden presentar errores o condiciones que no son requeridas para el Software, los registros ayudan al equipo de trabajo a realizar comparaciones donde se interpone un valor de fiabilidad y calidad entre el desarrollo, las evaluaciones y sus componentes.

Las pruebas de Software es un medio de comparación entre un resultado esperado y un resultado real, estos resultados se basan en la especificación de requerimientos y la ejecución de una prueba.

4.2 CARACTERÍSTICAS DE UNA PRUEBA DE SOFTWARE

Las pruebas de Software no se deberían de contemplar como una fase en los procesos o proyectos de Software, se debe considerar como herramienta continua de mejoramiento que se aplica a la par de un proceso, a continuación, se enlistan algunas de sus características.

- Una prueba de Software se debe realizar con la convicción de encontrar errores, al encontrar errores una prueba de Software se considera exitosa.
- Las pruebas de Software detectan diferentes tipos de errores.
- Los casos de prueba que se aplican para una prueba de Software deben ser diseñados para detectar errores que aún no son observados.
- Una prueba demuestra el cumplimiento de las especificaciones y requisitos a través de la evaluación de la funcionalidad del Software.
- El resultado de una prueba de Software demostrará la calidad que el Software presenta.

En la *Figura 4.1* se muestra los tópicos generales en los que se presentan las Pruebas de Software.

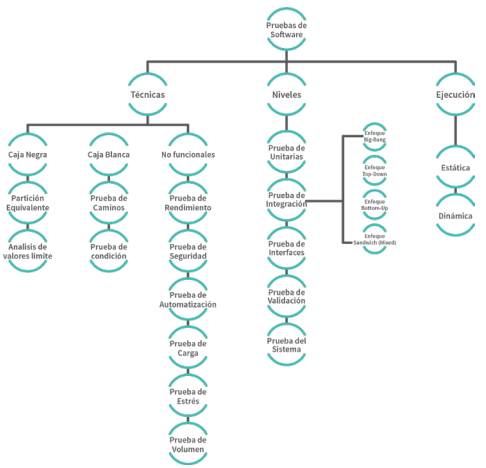


Figura 4. 1 Pruebas de Software

4.3 CICLO DE VIDA DE LAS PRUEBAS DE SOFTWARE

El ciclo de vida de las pruebas de Software (STLC) por sus siglas en Inglés, es un proceso donde se ejecuta una serie de actividades con el fin de asegurar que los objetivos de este proceso se han cumplido.

En la *Figura 4.2* se presenta el ciclo de vida del proceso de pruebas.

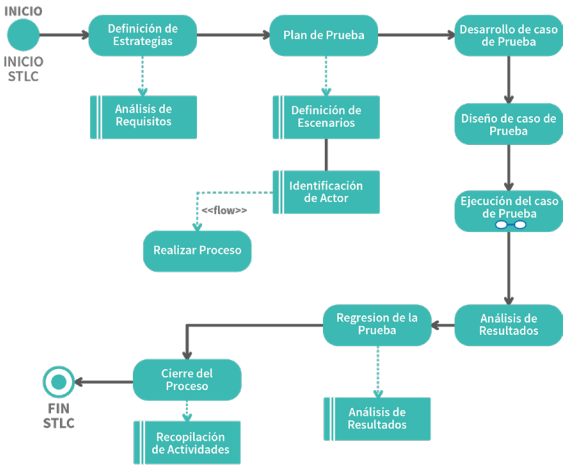


Figura 4. 2 Proceso de Prueba

4.3.1 DEFINICIÓN DE ESTRATEGIAS

Las definiciones en cuanto a las pruebas de Software se realizan a través de estrategias, las estrategias son las operaciones que se seguirán y los planes idóneos para dirigir un plan de pruebas, con el fin de alcanzar un objetivo y funcionamiento de un Software.

4.3.2 ANÁLISIS DE REQUERIMIENTOS

El análisis de requerimientos es un medio de comunicación entre el cliente y los integrantes del equipo. El análisis de requerimientos debe ser bien entendido, de aquí parte el inicio del proyecto, cuando los requisitos son bien entendidos el Software tendrá funciones explicitas bien hechas cubriendo las necesidades del cliente.

4.3.3 PLAN DE PRUEBA

El plan de prueba describe la estrategia que se usará para probar una aplicación, los recursos que se usarán, el entorno en el que se realizarán las pruebas, las limitaciones de las pruebas y la programación de las actividades de prueba. *[5]*

A continuación, se mencionan elementos importantes de una prueba de Software:

- Documentación donde se describe el plan de prueba.
- Objetos o entregables que serán Probados
- Casos de Prueba que se incluirán en los objetos que serán probados
- Características de las pruebas que serán utilizadas en el plan
  - Técnica
  - Nivel
  - Ejecución
- Recursos utilizados para la ejecución del plan.
- Riesgo involucrado al realizar el plan de prueba
- Cronograma de tareas

4.3.4 DEFINICIÓN DE ESCENARIOS

La definición de los escenarios son las circunstancias en las cuales se llevará a cabo el diseño de un caso de prueba, se determina el componente del Software que será probado, además se determinan las circunstancias o resultados esperados con base a la definición de un dato de entrada.

En la definición de casos de prueba intervienen:

- El tipo de actor que será capaz de realizar esta tarea específica, recordemos que los actores son identificados a través de los casos de uso del Proyecto de Software
- Tarea o proceso que se realizará.

4.3.5 DESARROLLO DE CASO DE PRUEBA

Mediante el diseño de un caso de prueba se genera el flujo se trabajó que deberá seguir una tarea específica en el Software.

Bob Binder (1991) *[6]* define un caso de Prueba como: "Un caso de prueba especifica el estado previo a la prueba y su entorno, las entradas o condiciones de prueba y el resultado esperado. El resultado esperado especifica qué se debe producir a partir de las entradas de la prueba. Esta especificación incluye los mensajes, las excepciones, los valores devueltos y el estado resultante y su entorno. "

En la *Figura 4.3* se observan las actividades del desarrollo de un caso de prueba

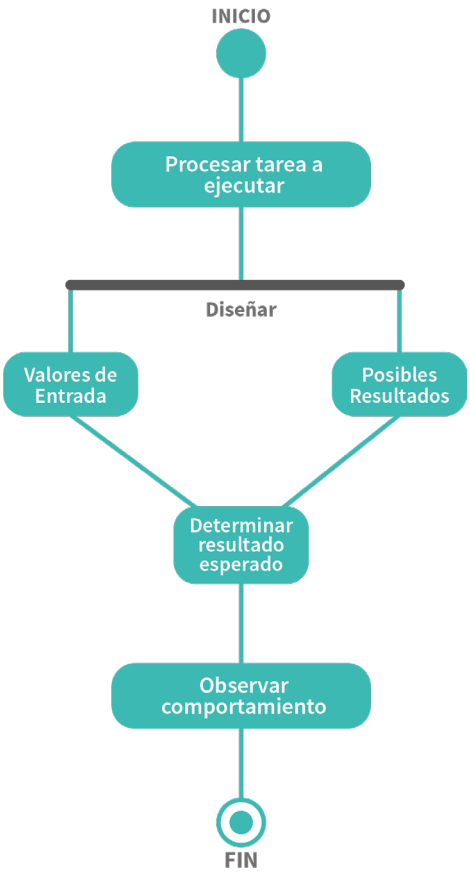


Figura 4. 3 Desarrollo del Caso de Prueba

4.3.6 EJECUCIÓN DEL CASO DE PRUEBA

Una vez diseñado el caso de Prueba se determinan las condiciones con las que será probado, se documenta los resultados, las observaciones realizadas y se identifican los posibles caminos o resultados de la prueba.

- Un escenario es el resultado esperado
- Una ruta alternativa que el usuario podrá seguir
- El escenario de error

4.3.7 ANÁLISIS DE RESULTADOS DE LA PRUEBA

Una vez realizada la ejecución del caso de Prueba se analizan los posibles resultados que se obtienen con el fin de determinar la reacción del Software antes de someterlo a producción.

4.3.8 REGRESIÓN DE LA PRUEBA

Se realiza un análisis de los resultados, se verifica la funcionalidad en los diferentes escenarios, cuando el Software es sometido a pruebas y se realizan modificaciones es necesario realizar este tipo de prueba para verificar las correcciones que se han hecho además de verificar que no se ha dañado o modificado ninguna funcionalidad previa a estas modificaciones.

4.3.9 CIERRE DEL PROCESO DE PRUEBA

Se recopilan todas las actividades, tareas, resultados observados, hechos que en las fases anteriores se obtuvieron para concluir todo lo realizado, esto se realiza a través de observaciones, discusiones y análisis del equipo de pruebas para identificar estrategias de implementación para mejorar el proyecto de Software que se está realizando. Este proceso se realiza con la intención de eliminar procesos que obstaculizan la calidad del Software.

4.4 CASO DE PRUEBA

IEEE define un caso de prueba como "Un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular, como para ejercer una ruta de programa particular o para verificar el cumplimiento con un requisito no específico" *[2]*

El caso de prueba hace referencia a los datos específicos que se utilizarán para probar una tarea específica, estos datos son llamados datos de entrada obteniendo un resultado esperado llamado datos de salida.

Para un caso de prueba debe de establecerse muy claro qué es lo que se quiere probar y comprobar, es necesario generar grandes procedimientos para un caso de prueba siempre y cuando sea el objetivo, recordemos que la prioridad en cada caso de prueba es detectar la mayor parte de errores.

Como sabemos existe infinidad de caminos o un conjunto de datos de entrada que generan un conjunto de resultados posibles, para mejorar este procedimiento se da la pauta para elegir de ese conjunto de prueba las mejores, por mejores nos referimos a que este caso de prueba observará detalles para el mejoramiento del proyecto de Software, todas estas *entradas* en un caso de prueba representan un dominio de entrada, ahora si hablamos de *dominio*, el *rango* son los posibles resultados al obtener la salida.

4.4.1 ERROR, DEFECTO, FALLA EN EL SOFTWARE

Al ejecutar un caso de prueba se puede observar el comportamiento del Software, una parte importante en el caso de prueba son las fallas, errores o defectos que un Software presenta cuando se ejecuta un caso de prueba. En la *Figura 4.4* se muestra la relación entre error, defecto falla

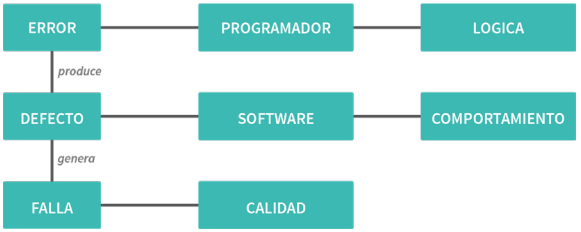


Figura 4. 4 Relación entre error, defecto, falla

4.4.1.1 ERROR

El error hace referencia a acciones humanas que generan resultados inesperados. La incomprensión de requisitos, error en evaluación de valores como tiempo y esfuerzo son algunas razones por las que se puede generar un error.

Estos son los tipos de errores que se pueden presentar en el Software: *[59]*

- ***Error Léxicos:*** producidos al escribir mal un identificador, una palabra clave o un operador.
- ***Error Sintácticos:*** por una expresión aritmética o paréntesis no equilibrados.
- ***Error Semánticos:*** como un operador aplicado a un operando incompatible.
- ***Error Lógicos:*** puede ser una llamada infinitamente recursiva.

4.4.1.2 DEFECTO

Es una manifestación en el Software realizada por una acción humana que no está bien ya sea en el código fuente, su diseño o proceso.

Los defectos se clasifican a continuación: *[58]*

- ***Muy alto:*** La imposibilidad de instalar / desinstalar el Software, el producto no se inicia, el producto se bloquea o el sistema operativo se congela, la corrupción de datos, el producto finaliza anormalmente, etc. por lo que también se denominan defectos catastróficos.
- ***Alto:*** La función no se ejecuta según las especificaciones, es crítica para el cliente, lo que significa que la aplicación puede continuar con defectos graves.
- ***Medio:*** Mensaje de error incorrectos, datos incorrectos, significa que la aplicación continúa con resultados inesperados.
- ***Bajo:*** Errores de gramática, son defectos con estas severidades son sugerencias dadas al cliente para mejorar la aplicación.

4.4.1.3 FALLA

Las fallas son observadas a través del comportamiento del Software, la manera de percibir una falla es a través de la ejecución del código.

4.4.2 OBJETIVOS DE UN CASO DE PRUEBA

Entre los objetivos de un caso de prueba se encuentra la detección de errores oportunos.

Al implementar casos de prueba entre los objetivos principales está el de encontrar errores que durante el proceso de creación del Software no se ha detectado. Lo importante es encontrar el mayor número de errores respecto al tiempo de creación del proyecto de Software.

Implementación de calidad para los módulos de Software.

Los casos de prueba hacen posible la madurez de un Proyecto de Software, no es posible brindar un proyecto de Software con errores, para entregar el proyecto es necesaria su observación, fiabilidad y validez.

Determinación de calidad entre cada proceso de creación de un proyecto de Software:

Para los testers y el equipo de trabajo, los casos de prueba ayudaran a la modificación y seguimiento del plan de trabajo, plan de pruebas, tareas específicas y el curso que toma el proyecto respecto a los objetivos que se necesitan alcanzar.

4.4.3 DISEÑO DE UN CASO DE PRUEBA

El diseño de un caso de prueba se realiza con la intención de obtener una probabilidad alta para descubrir errores, un caso de prueba no debe ser excesivo por lo contrario debe ser objetivo su contenido, debe tener el mejor plan de ejecución de pruebas que se aplicarán.

Un caso de prueba bien diseñado reduce los costos en tres categorías: *[33]*

- ***Productividad:*** tiempo para escribir y mantener los casos
- ***Capacidad de prueba:*** tiempo para ejecutarlos
- ***Fiabilidad:*** estimaciones efectivas.

El diseño de un caso de prueba consiste en elegir un conjunto de entradas de prueba bajo un resultado esperado, el objetivo es probar específicamente un objeto o atributo del Software. Para obtener un caso de prueba se necesita tener un propósito esto quiere decir que se tiene en claro que es lo que se probara, la forma o método en que la prueba se llevará a cabo y la versión de la aplicación de las pruebas.

Condiciones de ejecución en un caso de prueba *[32]* inventario de datos con los cuales se llevarán a cabo cada paso indicado en el caso de prueba.

Resultado esperado: reacción ideal que debe tener la aplicación ante un escenario y condiciones de ejecución indicadas. *[32]*

El diseño de un caso de prueba debe ser

- ***Correcto:*** es decir ser totalmente aprobado por el cliente y sus probadores.
- ***Objetivo:*** No debe de desperdiciar sus recursos respecto a su ejecución.
- ***Confiable:*** Obtener el mismo resultado cada vez que es ejecutado

4.5 TÉCNICAS DE LAS PRUEBAS DE SOFTWARE

Las pruebas de Software contienen técnicas de pruebas funcionales conocidas como "Caja Negra", estructurales "Caja Blanca" y pruebas no funcionales cada una divididas en diferentes tópicos, en las siguientes secciones se describe cada tópico que se utiliza para generar distintos casos de prueba a través de estas técnicas de prueba.

4.5.1 PRUEBA DE CAJA NEGRA

Esta prueba también es conocida como la prueba de entrada y salida, centrada en las especificaciones, esta prueba no se enfoca al procedimiento y a la lógica de codificación.

Este tipo de prueba se centra en la especificación de requerimientos, se observan las salidas de datos esperando que cumpla con los requerimientos y objetivos que anteriormente ya fueron descritos.

En la *Figura 4.5* se observa el esquema de comportamiento de una prueba de caja Negra.

Figura 4. 5 Prueba de Caja Negra

El componente o Software que está siendo evaluado se ve como una caja negra la única manera de evaluación es a través de entradas y salidas. La manera de observación es a través de la elección de un conjunto de datos específicos principalmente los datos que puedan generar errores.

Para estas pruebas se debe "encontrar una serie de datos de entrada cuya probabilidad de pertenecer al conjunto de entradas que causan dicho comportamiento erróneo sea lo más alto posible." *[34]*

El Software es probado a través de rutinas con pre y post condiciones, la categoría de errores que se presentan a realizar las pruebas de caja negra son errores de funciones incorrectas, por medio de las entradas y salidas de datos se pueden presentar errores de interfaz, de estructura de datos y de rendimiento.

4.5.1.1 PARTICIÓN EQUIVALENTE

Esta técnica de prueba permite dividir los datos de entrada y/o salida de un componente de Software para derivar el caso de prueba. La clasificación por datos de entrada y salida dependen del módulo que se esté probando ya que el comportamiento de las líneas de código miembros de una clase son similares.

La partición de equivalencia se esfuerza por definir un caso de prueba que descubra clases de errores, reduciendo así el número total de casos de prueba que se deben desarrollar. Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada. Una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica *[35]*

Las clases de equivalencia se definen de la siguiente manera *[35]*

- Si una condición de entrada especifica un rango, se definen una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada requiere un valor específico, se definen una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada especifica un miembro de un conjunto, se definen una clase de equivalencia válida y una no válida.
- Si una condición de entrada es booleana, se definen una clase válida y una inválida.

En la *Figura 4.6* se muestra el procedimiento para la partición de equivalencia.

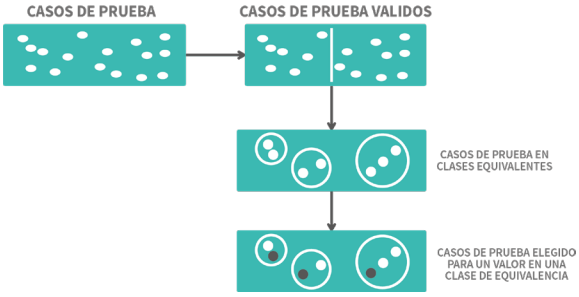


Figura 4. 6 Partición Equivalente

En la *Figura 4.6* se observa que se obtiene un conjunto de casos de prueba que posteriormente son elegidos como válidos e inválidos, posteriormente se agrupa los casos de prueba en clases equivalentes y por último se selecciona uno de los casos de prueba para un valor en una clase de equivalencia.

Como se observa cada clase de equivalencia contiene casos de prueba equivalentes, si existen distintos caminos de los cuales se puede obtener varios casos de prueba es necesario reducir el número de casos de prueba a uno.

4.5.1.2 ANÁLISIS DE VALORES LIMITE

Esta técnica es la más utilizada para las pruebas funcionales de Caja Negra. Los valores limites los encontramos en los valores que ponen a prueba el rendimiento, estos son los valores elevados, se presenta también cuando el valor de un índice especifico es nulo, la presencia de los valores máximos o negativos en bucles además de los valores no válidos para las variables.

El análisis del valor límite es una técnica de diseño de caso de prueba que complementa la partición de equivalencia. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el Análisis de Valores Limite conduce a la selección de casos de prueba en los "bordes" de la clase. *[35]*

A continuación, se presentan algunos aspectos importantes para diseñar casos de prueba con esta técnica. *[34]*

- Si una condición de entrada especifica un rango de valores, se diseñarán casos de prueba para los dos límites del rango, y otros dos casos para situaciones justo por debajo y por encima de los extremos.
- Si una condición de entrada especifica un número de valores, se diseñan dos casos de prueba para los valores mínimo y máximo, además de otros dos casos de prueba para valores justo por encima del máximo y justo por debajo del mínimo

4.5.2 PRUEBA DE CAJA BLANCA

En este punto se hace la evaluación interna del Software es decir de las líneas de código hechas por los programadores, un punto importante para realizar este tipo de prueba es conocer la lógica del Software. La ejecución de cada línea de código es importante para saber cómo responde cada línea, función o sentencia entre cada ejecución. Para esto se utilizan controladores que son llamadas a líneas de código que son evaluadas.

En la *Figura 4.7* se muestra el esquema de la prueba por caja blanca.

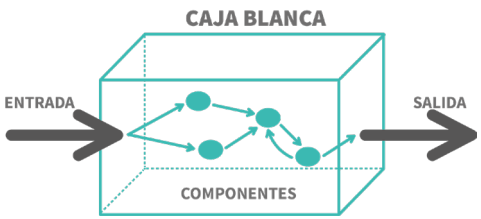


Figura 4. 7 Prueba de Caja Blanca

Las pruebas de caja blanca son pruebas que tienen en cuenta el mecanismo interno de un sistema o componente (IEEE, 1990). *[39]*

Un controlador es un módulo de software utilizado para invocar un módulo bajoprueba y, a menudo, proporcionar entradas de prueba, controlar y supervisar la ejecución e informar resultados de pruebas (IEEE, 1990) *[39]*

4.5.2.1 PRUEBA DE CAMINOS

Las pruebas de caminos son una estrategia de pruebas estructurales cuyo objetivo es probar cada camino de ejecución independiente en un componente o programa *[38]*

Se debe diseñar casos de prueba para probar los posibles caminos que un usuario realizaría para ejecutar una tarea específica. Por tanto, cada sentencia del código debe ejecutarse por lo menos una vez. Esta técnica se representa a través de grafos de flujo.

A continuación, se muestra las representaciones del grafo para esta técnica. *[34]*

- ***Nodos:*** Representan cero, una o varias sentencias en secuencia.
- ***Aristas:*** Flujo de control.
- ***Nodos predicados:*** Cuando en una condición aparecen uno o más operadores lógicos (AND, OR, XOR, etc) se crea un nodo distinto por cada una de las condiciones simples.

En la *Figura 4.8* se muestra un ejemplo de condición múltiple.

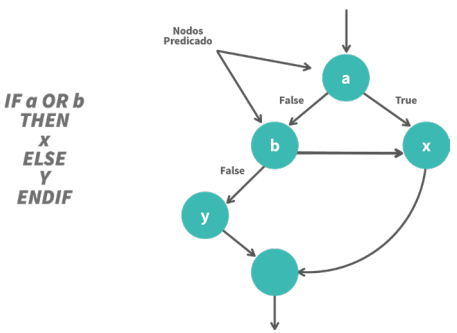


Figura 4. 8 Condición Múltiple [34]

4.5.2.2 PRUEBA DE CONDICIÓN

Se diseñan casos de prueba para que las sentencias del Software sean ejecutadas por lo menos una vez, el diseño debe incluir un resultado verdadero y otro falso.

La prueba de condiciones tiene como objetivo ejercitar adecuadamente todas las condiciones del módulo para buscar errores que puedan estar en variables lógicas, paréntesis, operadores relacionales o expresiones lógicas. *[37]*

Como se maneja y ejercitan condiciones lógicas para el diseño de casos de prueba utilizando esta técnica, se presentan distintos tipos de errores, estos se describen a continuación.

- ***Error por operador lógico (AND, OR, NOT):*** Se debe a su ausencia, al exceso o a la no pertenencia en las líneas de código.
- ***Paréntesis Lógico:*** La inclusión o exclusión de paréntesis puede cambiar el significado de algunos operadores.
- ***Error por operador relacional (<, <=, >,>=):*** Se debe a su ausencia, al exceso o a la no pertenencia en las líneas de código.
- ***Error por expresión aritmética.***

4.5.3 PRUEBAS NO FUNCIONALES

Estas pruebas se enfatizan en la evaluación de evaluar el Software a través de los requisitos no funcionales, es decir se enfoca al funcionamiento y utilización del Software, y no por casos de prueba que especifican su comportamiento.

4.5.3.1 PRUEBA DE RENDIMIENTO

La prueba de rendimiento es el proceso mediante el cual se prueba el software para determinar el rendimiento actual del sistema. Este proceso tiene como objetivo recopilar información sobre el rendimiento actual, pero no establece juicios de valor sobre los resultados. *[41]*

Esta prueba determina la aceptación de acceso y tiempos de respuesta para el Software, se hace referencia al uso del Software y su funcionamiento, algunos puntos que se evalúan para el Software son el tiempo de respuesta, el procesamiento de datos (throughput), la entrada de datos para procesar una tarea específica e ingresar a otro modulo del Software a través de su interfaz.

Existen tres actividades importantes que se realiza para este tipo de Prueba: *[42]*

- Ejecución de la prueba de rendimiento para determinar los parámetros de rendimiento dentro del Software.
- Análisis de los resultados para examinar si el Software está cumpliendo los objetivos de rendimiento o no.
- Optimización del Software para resolver los cuellos de botella de rendimiento

4.5.3.2 PRUEBA DE SEGURIDAD

La prueba de seguridad es una técnica de prueba para determinar si un sistema de información protege los datos y mantiene la funcionalidad según lo previsto. *[43]*

Bajo estas pruebas se observa la vulnerabilidad del Software, debe existir un control de acceso para los usuarios, algunos atributos para este tipo de pruebas deben reflejar la integridad de los datos, autenticación, autorización y disponibilidad.

Algunos beneficios que proporcionan estas pruebas son:

- Proteger la información de destinatarios erróneos
- Proporciona la integridad de datos, es decir, se realizan transacciones de información a módulos del Software correctos.
- Identificación correcta de los tipos de datos y acceso a módulos que y recursos que le pertenece.
- Disponibilidad de recursos y de información en todo momento.

4.5.3.3 PRUEBA DE AUTOMATIZACIÓN

La automatización de las pruebas funcionales reduce significativamente el esfuerzo dedicado a las pruebas de regresión en productos que se encuentran en continuo mantenimiento. *[44]*

Las pruebas de automatización mejoran las expectativas y proporciona una mayor cobertura del Software que se está probando, permite a los testers obtener una mayor rapidez respecto a los datos que conforman el Software,

Las pruebas de automatización requieren una cantidad considerable de inversión para comprar el software y los recursos de hardware compatibles. Las pruebas de automatización mejoran la precisión y ahorran tiempo al dinero del tester y la organización. *[45]*

Este tipo de prueba se presenta cuando los requisitos por parte del cliente cambian constantemente y es necesario modificarlas o en otro caso el caso de prueba es utilizado en múltiples ocasiones.

4.5.3.4 PRUEBA DE CARGA

La prueba de carga se realiza para probar el comportamiento de la aplicación en varios niveles de carga dentro de sus límites aceptables. El parámetro principal para enfocar durante la prueba de carga es *"Tiempo de respuesta".[42]*

Esta prueba también es conocida como prueba de esfuerzo, esta prueba evalúa al Software respecto a un gran número de datos y el tiempo de respuesta que se obtiene para realizar una tarea.

Bajo esta prueba se observa el constante comportamiento de una transacción específica, se comprueba el nivel de aceptación por parte de los usuarios y el momento en que el Software empieza a fallar.

Existen diferentes tipos de Prueba de Carga:

- ***Prueba de carga a nivel componente:*** Acceso múltiple por parte de usuarios a un módulo, mide los interbloqueos, pérdidas de memoria, rendimiento y sincronización.
- ***Prueba de carga de Infraestructural:*** Mide los recursos con los que cuenta el Software como rendimiento, escalabilidad, además de que observa las configuraciones para ajustar el Software a un escenario específico.
- ***Prueba de carga Arquitectónica:*** Observa a el Software bajo diferentes niveles, en este punto el Software es visto como un todo, pero se verifica la funcionalidad de este en diferentes circunstancias.

4.5.3.5 PRUEBA DE ESTRÉS

La prueba de estrés es una técnica de gestión de riesgos utilizada para evaluar los efectos potenciales, de un conjunto de cambios especificados en los factores de riesgo, correspondientes a eventos excepcionales pero plausibles. *[47]*

Busca encontrar el procesamiento incorrecto dentro del Software, a través de este caso de prueba se ejecutan casos de prueba en el cual múltiples usuarios entran al Software paralelamente para observar el comportamiento del Software, los usuarios hacen uso de este módulo, de los mismos datos de entrada y realizan una tarea específica durante un corto tiempo. Para este análisis se observa que no exista problemas en entender la interfaz, el tiempo de respuesta y la generación de datos que se van produciendo con su respectivo resultado.

A continuación, se observan algunas pruebas de Estrés: *[46]*

- ***Sistema en Línea:*** Ingreso al Software, observación de tiempo en uso.
- ***Sistema de Base de Datos:*** Número máximo de usuarios conectados. Números máximos de transacciones paralelas, incluso contra la misma tabla o elemento de datos.
- ***Intercambio de Archivos:*** esta prueba sólo es interesante si los archivos se envían en paralelo a través de una red. Probar el envío desde cada canal al mismo tiempo y verificar colisiones, tiempos de espera o transferencias incompletas

4.5.3.6 PRUEBA DE VOLUMEN

Este Tipo de prueba no funcional verifica el Software a través de un extenso contenido de datos tanto de entrada y/o salida. Se elijen distintos tipos de datos para iniciar con las pruebas observando el resultado conforme a la interacción del Software y la base de datos.

Uno de los objetivos primordiales de este tipo de prueba no funcional es verificar el punto en el que un Software empieza a fallar cuando una cantidad razonable de datos son buscados, utilizados u ordenados. Como se observa este tipo de pruebas se realiza y observa con la interfaz del Software y su interacción con la base de datos.

Parte de la prueba es ejecutar el sistema durante un tiempo determinado con una gran cantidad de datos. Esto está en orden para verificar qué ocurre con los búferes temporales y los tiempos de espera debido a los largos tiempos de acceso. *[48]*

4.6 NIVELES DE PRUEBA DE SOFTWARE

El acto de diseñar pruebas es conocido como una de las mejores actividades de prevención de errores. El diseño de las pruebas puede descubrir y eliminar errores en cada etapa en el proceso de construcción del software (*Beizer* 1990).

Los niveles de prueba dependerán de la fase del proyecto en la que nos encontremos, entre los niveles de prueba encontramos:

- Pruebas Unitarias
- Pruebas de Integración
- Pruebas de Interfaces
- Pruebas de Validación
- Pruebas del Sistema

Generalmente en los niveles de prueba se inicia por componentes y módulos del Software para que posteriormente se inicie con la integración de estos módulos viendo al Software como un todo. Partimos desde los módulos del Software verificando su código, después se analiza el diseño del Software generando la Integración y al final se hace la comparación del cumplimiento de los requisitos del cliente con el Software.

En la *Figura 4.9* se muestra la representación de los niveles de prueba.

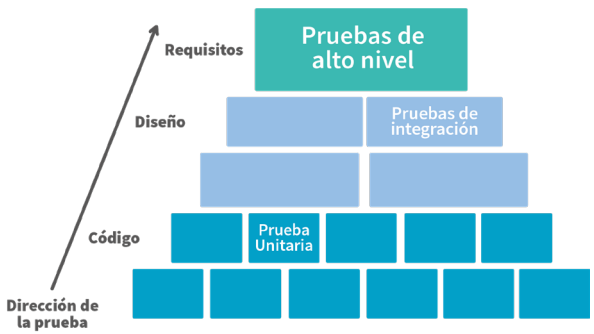


Figura 4. 9 Niveles de Prueba de Software [49]

4.6.1 PRUEBAS UNITARIAS O DE COMPONENTE

"Un módulo [componente] debe ser abierto para la extensión, pero cerrado para la modificación" [16]

La prueba de unidad enfoca los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de Software. *[16]*

Las pruebas unitarias, también llamadas pruebas de componentes, se encargan de probar, individualmente, subprogramas, subrutinas o procedimientos en un programa. El propósito de la prueba es comparar la función de un módulo con alguna especificación que defina el módulo. *[38]*

Este tipo de prueba se realiza para cada componente del Software de manera Independiente. Se analiza la lógica del módulo en el que se trabaja garantizando que se cumpla con las especificaciones del Software, además se verifica que la nomenclatura estándar del código sea correcta (nombre de clases, nombre de funciones, tipos de parámetros, etc.).

Permite al equipo de testers analizar el comportamiento de módulos tratados de manera independiente a través de datos de entrada y flujos de control con el fin de verificar el comportamiento de las funciones en el módulo. Algunos resultados se presentan como erróneo, inesperado o exitoso.

Cuando se termina la codificación del módulo, el tester realiza las pruebas de unidad, el proceso de liberación se genera hasta que el tester considera que el módulo está libre de errores. Este nivel sugiere que cada componente que de prueba tenga como mínimo dos casos de prueba diseñados para ellos, el primero consiste en un caso de prueba exitoso y el segundo es un caso de prueba que arroje resultados erróneos.

Al no realizar este nivel de prueba, se generan más errores, se obliga al equipo de desarrollo a trabajar en la corrección de errores de múltiples líneas de código, el resultado de esto es el incremento en tiempo de desarrollo.

4.6.1.1 ¿QUÉ ES UNA UNIDAD DE SOFTWARE?

Un componente es una entidad separada, un objeto, en definitiva, que se desarrolla y se prueba como una unidad con independencia del contexto en el que será utilizado finalmente. *(Manish B. Mehta) [2]*

Existen diferentes tipos de componentes que pueden probarse en este nivel de prueba *[38]*

- Funciones individuales o métodos dentro de un objeto.
- Clases de objetos que tienen varios atributos y métodos.
- Componentes compuestos formados por diferentes objetos o funciones. Estos componentes compuestos tienen una interfaz definida que se utiliza para acceder a su funcionalidad.

En la *Figura 4.10* se muestra las estrategias de una prueba unitaria.

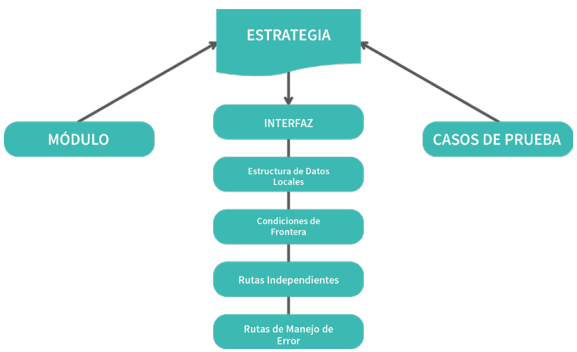


Figura 4. 10 Estrategia de Prueba Unitarias

Estrategias para Casos de Prueba en pruebas Unitarias [16]

- *Interfaz:* La interfaz del módulo se prueba para garantizar que la información fluya de manera adecuada hacia y desde la unidad de software que se está probando.
- *Las Estructuras de Datos Locales:* se examinan para asegurar que los datos almacenados temporalmente mantienen su integridad durante todos los pasos en la ejecución de un algoritmo.
- *Las Condiciones de Frontera* se prueban para asegurar que el módulo opera adecuadamente en las fronteras establecidas para limitar o restringir el procesamiento.
- *Rutas Independientes:* Asegura que todas las sentencias de la unidad se ejecutan por lo menos una vez.
- *Rutas de Manejo de Error:* Si los datos no entran y salen de manera adecuada, todas las demás pruebas son irrelevantes.

4.6.2 PRUEBAS DE INTEGRACIÓN

Se realizan las pruebas de cada uno de los componentes del Software que interactúan entre sí, entendiendo la lógica del Software. En este proceso se unen los módulos que ya fueron probados y después se vuelven a probar.

Se realiza la combinación de módulos que ya pasaron por el proceso de pruebas, se asegura la interacción, comunicación y los datos que darán lugar a la comunicación entre interfaces y enlaces del Software.

Las pruebas de integración son una técnica sistemática para construir la arquitectura del software mientras se llevan a cabo pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por diseño. *[16]*

4.6.2.1 ESTRATEGIAS PARA CASOS DE PRUEBA EN PRUEBAS DE INTEGRACIÓN

- Integración Big Bang: El sistema es tratado como un subsistema, se prueba en una sola fase.

Este enfoque requiere pocos recursos para ejecutarse ya que no necesitamos identificar los componentes críticos ni exigir una codificación adicional para los “módulos ficticios”. [50] En la *Figura 4.11* se muestra la representación de la estrategia de Big- Bang.

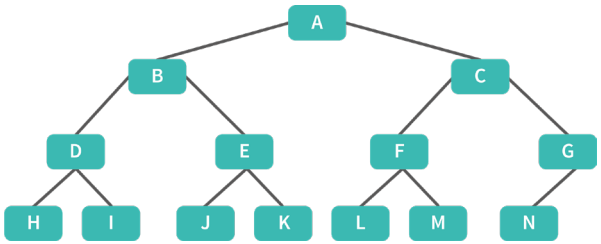


Figura 4. 11 Integración Big Bag

■ ***Integración de arriba hacia abajo (top-down):*** En la integración de arriba hacia abajo, comenzamos en el nodo de destino en la raíz del árbol de descomposición funcional y trabajamos hacia las hojas. *[50]*

Esta Integración ocurre desde el módulo o programa principal del Software, posteriormente los subprogramas se van incorporando, la manera en cómo se incorpora dependerá de las características específicas con las que el Software va necesitando.

Para esta estrategia existen dos tipos de integración, la integración por profundidad esta se representaría integrando los componentes de mayor prioridad del Software. Ejemplos si se selecciona una ruta por la izquierda se seleccionarían los componentes A, B después se integraría el componente D y posteriormente los componentes H o I. En la *Figura 4.12* se muestra la Integración por Profundidad

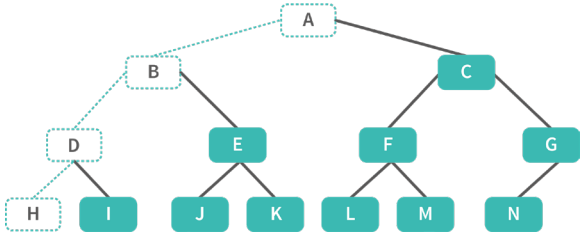


Figura 4. 12 Integración por profundidad

■ ***La integración por anchura*** incorpora los subprogramas por nivel, la integración sucede horizontalmente, como ejemplo se inicia la integración de los módulos B Y C posteriormente D, E, F, G. En la *Figura 4.13* se muestra la integración por anchura.

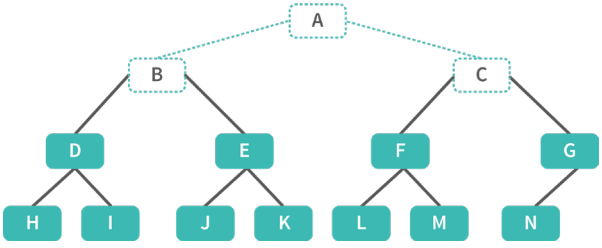


Figura 4. 13 Integración por Anchura

Pressman Indica en proceso de Integración top-down de la siguiente manera: *[16]*

- El módulo de control principal se usa como un controlador de prueba y los representantes se sustituyen con todos los componentes directamente subordinados al módulo de control principal.
- Dependiendo del enfoque de integración seleccionado los representantes subordinados se sustituyen uno a la vez con componentes reales.
- Las pruebas se llevan a cabo conforme se integra cada componente.
- Al completar cada conjunto de pruebas, otro representante se sustituye con el componente real.
- Las pruebas de regresión para asegurar que no se introdujeron nuevos errores.

■ ***Integración ascendente (Bottom-Up):*** La integración de los módulos de Software para esta estrategia comienzan en los componentes inferiores hacia el componente principal del Software.

Este enfoque permite comenzar a trabajar con un nivel de implementación más simple e inferior, lo que permite crear entornos de prueba más fácilmente debido a las salidas más simples de esos módulos. La integración ascendente se usa comúnmente para sistemas orientados a objetos, sistemas en tiempo real y sistemas con requisitos estrictos de rendimiento [50] En la *Figura 4.14* se muestra la integración ascendente.

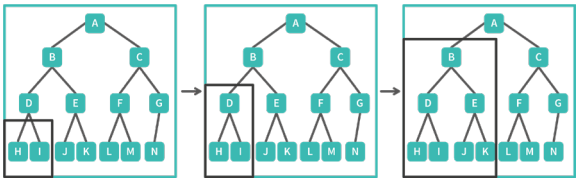


Figura 4. 14 Integración Ascendente

- *Integración Sándwich*

La integración de Sándwich utiliza un enfoque mixto donde utilizamos stubs en el nivel superior del árbol y controladores en el nivel inferior. La dirección de prueba comienza desde ambos lados del árbol y converge hacia el centro, por lo tanto, el término sándwich. *[50]*

Este tipo de integración une tanto la integración de arriba hacia abajo (top-down) y la integración ascendente (Bottom-Up), se utilizan las ventajas de ambas estrategias de integración ya que una se inicia por el programa principal que maneja a todos los módulos y por el contrario la otra inicia desde los módulos internos, se generan casos de prueba que generan la cobertura total del Software. *En la Figura 4.15* se muestra la Integración por Sándwich.

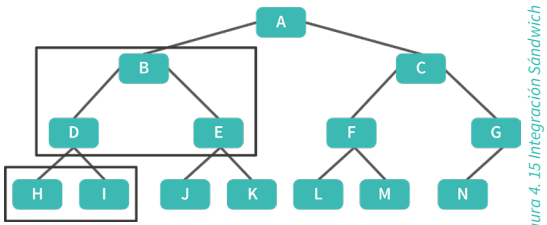


Figura 4. 15 Integración Sándwich

4.6.3 PRUEBAS DE INTERFACES

Las interfaces gráficas de usuario dividen las acciones estándar del usuario y los paradigmas de trabajo en varias categorías que se presentan gráficamente de manera que reflejan su funcionalidad. *[52]*

Las pruebas de interfaces se deben organizar como una jerarquía de scripts, es decir, archivos que contienen comandos para simular las acciones del usuario y verificar los resultados. *[52]*

Las prioridades para estas pruebas es obtener el buen funcionamiento en las Interfaces que conforman el proyecto de Software, las pruebas que se realizan son a través de los comandos que para cada módulo fue diseñado, la validez de campos obligatorios, la transición entre interfaces, valores predeterminados, validación de datos en formularios, manejo de errores que conforman los mensajes que visualizará el usuario y tiempo y resultados obtenidos por consultas.

Estas son algunas de las estrategias que son implementadas para realizar pruebas de interfaces. *[51]*

- Se implementan casos de prueba donde se describen tareas básicas que un usuario final debe realizar dentro del Software.
- Las tareas y mensajes de la interfaz son comparadas con los objetivos y la capacidad de aprendizaje de un usuario, se observan las discrepancias entre las expectativas del usuario y las actividades que realiza el usuario para realizar la tarea.
- Las pruebas de usabilidad se realizan dentro del proyecto de Software se estudia en condiciones reales o controladas (usuarios reales), y los evaluadores recopilan datos sobre los problemas que surgen durante su uso.

Estas estrategias se implementan mediante la descripción y elección del tipo de entrada para el caso de prueba que será utilizado en la interfaz a probar, ahora se inicia con la elección de los tipos de parámetro para la entrada en este punto se observa si los parámetros elegidos pueden ser útiles para otros casos de entrada.

Con la elección de parámetros se evaluarán los eventos entre cada entrada y la relación entre otras interfaces.

- Cada acción está asociada con algunos eventos, y cada evento genera un nuevo estado. *[53]*

La automatización de este tipo de pruebas es esencial para cubrir la mayoría de las tareas y la comunicación entre todo el Software en sus Interfaces.

4.6.4 PRUEBAS DE VALIDACIÓN

Esta prueba se realiza al culminar las pruebas de integración y de interfaces, ya que el Software es considerado como un todo.

La validación se puede definir de muchas maneras, pero una definición simple es que la validación tiene éxito cuando el software funciona de una manera razonablemente y se adecua a los requisitos esperados por el cliente. La validación del software se logra a través de una serie de pruebas de caja negra que demuestran conformidad con los requisitos. *[35]*

En este punto los casos de prueba se debe especificar que la funcionalidad del Software se apega o cumple los requisitos funcionales antes descritos, mediante la validación se observa la mayoría de los factores de calidad que en el Capítulo 2 se describe.

Como se observa existen dos tipos de resultados:

- Los resultados de la prueba demuestran que el Software se apega a los requisitos y es aceptada.
- Entre los requisitos y el Software que se está probando existen objetos, módulos, funcionalidades o componentes del Software que no están en el requisito por lo tanto es rechazado.

Cuando un equipo de trabajo realiza un proyecto de Software donde se implementa las prueban de validación deben de existir pruebas de aceptación, el problema recae en que el proceso de aceptación tarda mucho y pasa por muchos ciclos para ser aceptado es por lo que se implementaron “Pruebas Alfa” y “Pruebas Beta” para descubrir errores que usuarios finales pudieran detectar al momento de utilizar el Software.

4.6.4.1 PRUEBAS ALFA

Este tipo de pruebas se realizan en el entorno de trabajo del desarrollo, a través de ella los evaluadores observan a los usuarios evaluando el desempeño del usuario al usar el Software. Este tipo de prueba es importante para obtener la calidad en el Software creado, esto se debe a que si el resultado arrojó algún cambio o error presente en el Software al momento de su utilización las modificaciones se pueden realizar al momento.

Las pruebas alfa son pruebas finales antes de entregar el Software a su utilización o producción final. En la *Figura 4.16* se muestra el proceso de pruebas Alpha.

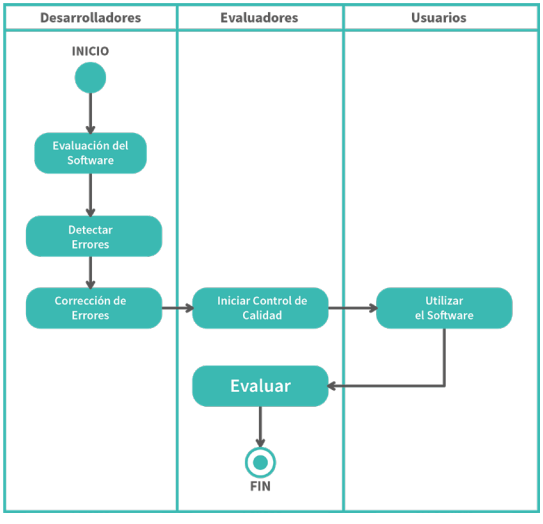


Figura 4. 16 Prueba Alpha

4.6.4.2 PRUEBAS BETA

Este tipo de pruebas se implementan en el contexto de producción del Software, el Software es instalado y se evalúa bajo condiciones reales de trabajo. Al iniciar la evaluación en un contexto real permite a los usuarios tener un acercamiento al Software, se pueden presentar errores que permiten al equipo de trabajo detectarlos antes de su versión final.

Existen dos tipos de pruebas Beta:

- *Las versiones beta cerradas:* Se aplican a un grupo específico de usuarios.
- *Las versiones beta abiertas:* Se aplican a usuarios interesados en el Software.

4.6.5 PRUEBAS DEL SISTEMA

Las pruebas del sistema prueban el Software como un todo. Una vez que todos los componentes están integrados, el Software se prueba rigurosamente para verificar que cumple con los estándares de calidad específicos. *[54]*

La especificación de requisitos es un elemento importante para este tipo de prueba, la planificación de los casos de prueba otro factor porque se realizarán los casos de prueba que permitan realizar la evaluación de todo el Software, esto se hace a través de los niveles de Prueba antes mencionados.

Para estas pruebas influye la especificación de requisitos, el plan de prueba, el diseño de caso de prueba la implementación de niveles y los factores presentes en cada componente del Software. Aquí influye la calidad con la que se realizó el proceso de pruebas porque reflejan el estado especifico del Software a través de las evaluaciones de sus componentes y las pruebas realizadas en ellos.

4.7 EJECUCIÓN DE LAS PRUEBAS DE SOFTWARE

La ejecución de las pruebas de Software se refiere al modo en que una prueba de Software se va a utilizar para probar el Software. La ejecución de las pruebas ayudara al proceso de Software a entregar un proyecto de calidad.

La detección de fallas del software proporciona evidencia si el comportamiento de un programa cumple con lo especificado durante la ejecución del programa. *[52]*

4.7.1 EJECUCIÓN ESTÁTICA

Las técnicas estáticas se relacionan con el análisis y la verificación de las representaciones del sistema como: la documentación de requisitos, los diagramas de diseño y el código fuente del programa, de forma manual sin ejecutar el código *[55]*

Las actividades que se realizan manualmente se relacionan con la inspección y detección de errores oportunos en la parte de diseño esto se realiza con el fin de mejorar la calidad del Software en las primeras fases del diseño de Software.

La ventaja de una ejecución estática es que al detectar defectos en las primeras etapas de desarrollo y diseño del Software el costo de modificación por cada error es mínimo.

Las técnicas estáticas se pueden agrupar en dos categorías:

- *Revisión*

  Básicamente, una revisión es una variedad de actividades que implican la evaluación de la materia técnica por un grupo de personas que trabajan juntas *[51]*

- *Análisis*

  El análisis estático es una técnica de detección de fallas que evalúa un sistema o componente basado en su forma, estructura, contenido o documentación. Estático análisis se utiliza para controlar si un objeto bajo el análisis cumple con las expectativas uniformes en torno a la seguridad, confiabilidad, rendimiento y capacidad de mantenimiento. *[51]*

En la Figura 4.17 se puede observar la ejecución estática.

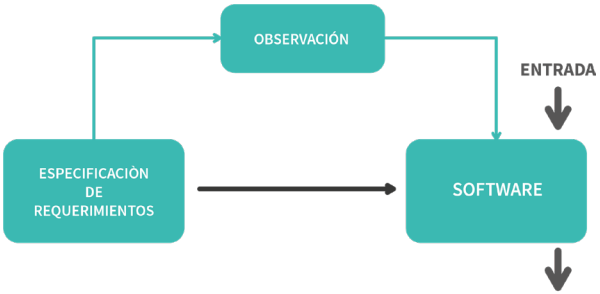


Figura 4. 17 Ejecución Estática

4.7.2 EJECUCIÓN DINÁMICA

La ejecución dinámica es un conjunto de técnicas para examinar rigurosamente un programa en función de algunos criterios durante el tiempo de ejecución, el análisis dinámico es muy adecuado para realizar evaluaciones basadas en el tiempo de ejecución o ejecución del programa. *[52]*

Algunos criterios en los cuales se enfoca esta ejecución es en el análisis de cobertura del código, especificación de criterios de salida para compararla con los requisitos del sistema e implementación de pruebas.

Mediante esta ejecución el módulo es visto por módulos donde se realizan ejecuciones constantes y se realizan historiales donde se anotan las observaciones del comportamiento. Esta ejecución es precisa porque se sabe cuál es el resultado que se desea obtener a través de las ejecuciones.

Este tipo de análisis puede examinar el comportamiento real y exacto del tiempo de ejecución de un programa. *[57]*
En la *Figura 4.18* se muestra la ejecución dinámica.

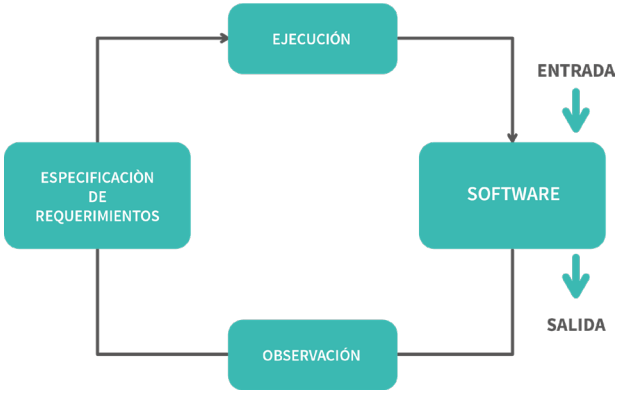
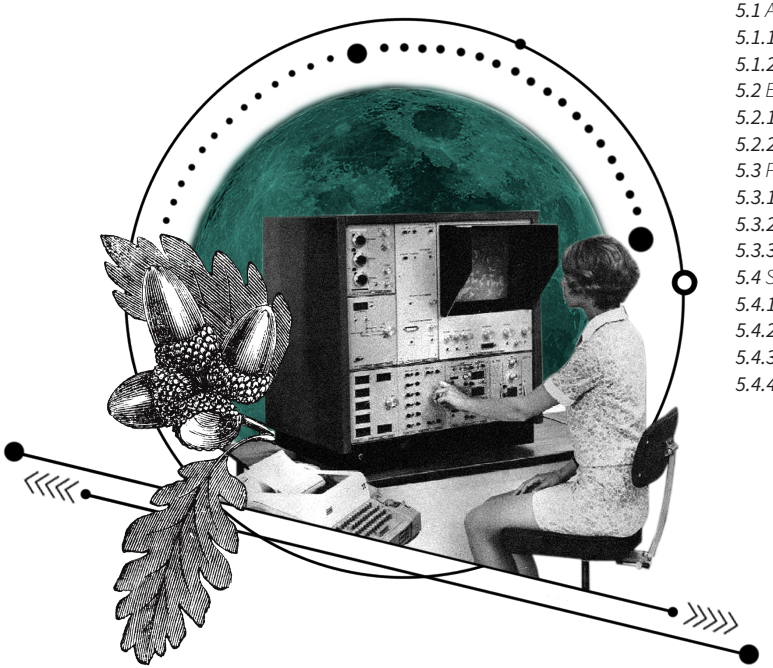


Figura 4. 18 Ejecución Dinámica

CAPÍTULO 5

PRUEBAS DE SOFTWARE A TRAVÉS DE UN FRAMEWORK



PRUEBAS DE SOFTWARE A TRAVÉS DE UN FRAMEWORK

En esta investigación se han descrito algunas ineficiencias que se pueden presentar en el Software provocadas por los desarrolladores, errores humanos, esta es una de las razones por las cuales la automatización de las pruebas se ve como un beneficio ya que al utilizar un Software de automatización se reduce la probabilidad de generar errores humanos y se aumenta la eficiencia.

La automatización de las pruebas de Software ayuda a reducir el tiempo de la prueba y sus costos. Las pruebas son determinadas como elementos que serán probados a través de un Software. Este Software es un framework, los casos de prueba en este caso son escritos en código del framework que se encargara de ejecutar las pruebas.

La automatización de pruebas se ayuda de frameworks para simular otras partes del Software. La automatización de las pruebas pretende realizar pruebas que se llevarían mucho tiempo si se realizaran manualmente, o también para las pruebas que son iterativas, el control y la comparación de resultados previstos son a través de la automatización.

Los casos de prueba son secciones de código que se probaran para saber si el Software cumple con las especificaciones y actúan de una manera esperada bajo ciertas circunstancias.

Otra forma de automatización de Software es a través de su interfaz, existen frameworks donde se observa la interacción de los usuarios, estas herramientas graban el comportamiento de los usuarios a través de la interfaz para que posteriormente se analice el resultado de la prueba con el resultado esperado.

5.1 AUTOMATIZACIÓN DE PRUEBAS DE SOFTWARE

La automatización es la integración de herramientas de prueba de tal manera que la ejecución de la prueba, el registro, y la comparación de los resultados se realizan con poca intervención humana. ***(Koirala y Sheikh, 2009)***

La automatización de pruebas se realiza a través de un Software especial que controla la ejecución de las pruebas. En este punto hablamos de una reducción considerable de tiempo y costo. Reduce la carga de trabajo ya que al ser un Software la ejecución y verificación se puede realizar en diferentes momentos.

En la automatización de las pruebas se siguen casos de prueba específicos, esto quiere decir que se sigue una secuencia especifica de código que será ejecutada con precisión.

Según SmartBear, la automatización de las pruebas de Software son la mejor manera de aumentar la eficacia, la eficiencia y la cobertura de las pruebas de software. La automatización de pruebas pueden conducir a una mejora sustancial en el proceso de desarrollo y también simplifica el soporte y mantenimiento de Software. *[10]*

Existen dos estrategias de prueba:

- *Prueba basada en código.*

  Este tipo de pruebas se realiza a través de la generación de datos de entrada, posteriormente estos datos se implementan en el caso de prueba que será ingresado al framework para observar su resultado. Mediante el framework las pruebas de Software son a través de unidades que son evaluadas en diferentes circunstancias.

- *Prueba grafica de la interfaz de usuario.*

  Se generan casos de prueba que consisten en describir una tarea específica dentro del Software, se observan los diferentes eventos y las rutas que el usuario utiliza para realizar esta tarea específica. A través de la evaluación de estas observaciones se puede determinar el éxito o fracaso del caso de prueba.

5.1.1 BANCO DE PRUEBAS

Un banco de pruebas del software es un conjunto integrado de herramientas para soportar el proceso de pruebas. *[38]*

- ***Gestor de pruebas:*** Mantiene un registro de los datos de las pruebas, resultados esperados y facilidades del programa que han sido probadas.
- ***Generador de datos de prueba.*** Genera datos de prueba para el programa a probar, para generar datos aleatorios de forma correcta.
- ***Oráculo.*** Genera predicciones de resultados esperados de pruebas.
- ***Comparador de ficheros.*** Compara los resultados de las pruebas del programa con los resultados de pruebas previos e informa de las diferencias entre ellos.
- ***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.

5.1.2 GENERACIÓN DE DATOS DE PRUEBA

Como sabemos la generación de datos en una prueba de Software es indispensable esto se debe a que, por medio de los casos de prueba se eligen los datos que se utilizarán para implementar y evaluar la calidad de la prueba de Software a través de la ejecución del caso de prueba.

La generación automatizada de datos de prueba (ATDG) es una actividad que en el transcurso de las pruebas de software genera automáticamente datos de prueba para el software bajo prueba (SUT). Hace pruebas de software más eficiente y rentable. *[20]*

Muchos generadores de datos de prueba automatizados se basan en la ejecución simbólica. La ejecución simbólica proporciona una representación funcional de la ruta en un Software y asigna variables simbólicas para los valores de entrada y evalúa una ruta interpretando las declaraciones y predicados

en la ruta en términos simbólicos. Este es un enfoque estático de automatización de pruebas.

El enfoque dinámico se ejecuta a través de SUT, se inserta la entrada y se observa la trascendencia de la entrada con el fin de que la entrada tome una ruta esperada. Cuando el dato de entrada recorre una ruta no deseada se retoma la ruta desde el punto en donde fallo el dato.

En la *Figura 5.1* se muestra el proceso que tiene un generador de datos de prueba.

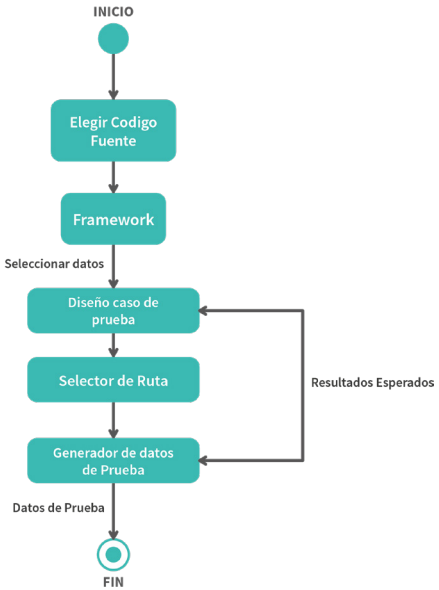


Figura 5. 1 Generador de datos de prueba

La elección de código es la parte del Software que se analizará, a través del framework se obtendrá información precisa que se genera para una prueba, en este punto encontrar los valores correctos es una tarea difícil porque se seleccionará una ruta que obtendrá un camino específico y obtener un resultado que se espera a través de datos de prueba.

5.1.2.1 ESTRATEGIA DE GENERACIÓN DE DATOS DE PRUEBA ORIENTADO A OBJETIVOS

Korel lo define como el proceso de generar datos de prueba de entrada para ejecutar la instrucción seleccionada independientemente de la ruta tomada, es decir, se elimina la etapa de selección de ruta. *[22]*

Para esta estrategia los datos de entrada son ejecutados a través de una ruta especifica de prueba.

5.1.2.2 ESTRATEGIA DE GENERACIÓN DE DATOS DE PRUEBA ORIENTADO A RUTA

Para esta estrategia se obtienen múltiples caminos por el cual el dato de entrada viaja, para esta estrategia sólo se elige uno, cuando se selecciona la ruta se observa si la entrada es correcta en conjunto con sus datos.

Esta estrategia contiene 3 pasos importantes

- Analizador de Programas
- Selector de Rutas
- Generador de Datos de Prueba

5.1.2.3 ESTRATEGIA DE GENERACIÓN DE DATOS DE PRUEBA ALEATORIA

Como su nombre lo indica con esta estrategia se toman y generan datos aleatorios hasta encontrar datos óptimos para ser ocupados. Esta estrategia es un tanto confusa, porque existe un dominio de datos enorme para un caso de prueba, lo primordial para esta estrategia es la elección de datos que se acoplen a una ruta especifica.

La prueba aleatoria es el proceso de selección de la prueba al azar de acuerdo con una distribución de probabilidad uniforme sobre el dominio de entrada de programas. *[22]*

5.2 ECLIPSE

Es un IDE de código abierto multiplataforma de desarrollo. Es un entorno de desarrollo integrados obtiene un modelo de capas por medio de los puntos de extensión, estos permiten la interconexión de los plugin ofreciendo una interfaz y procesar resultados. La calidad de la experiencia del usuario depende significativamente de qué tan bien se integran las herramientas con la Plataforma y de qué tan bien las herramientas trabajan entre sí.

La interfaz de usuario de la plataforma Eclipse está construida alrededor del workbench que proporciona la estructura general y presenta un interfaz de usuario extensible para el usuario. La API y la implementación del workbench se crean a partir de dos kits de herramientas: *[29]*

- ***SWT:*** un conjunto de widgets y una biblioteca de gráficos integrados con el sistema de ventanas nativo, pero con un API independiente del sistema operativo.

- ***JFace:*** un conjunto de herramientas de interfaz de usuario implementado con SWT que simplifica las tareas de programación de IU comunes.

5.2.1 CARACTERÍSTICAS DE ECLIPSE

Eclipse tiene las siguientes características: [29]

- Apoya la construcción de herramientas para el desarrollo de aplicaciones.

- Herramientas de soporte para manipular tipos de contenido arbitrarios (por ejemplo, HTML, Java, C, JSP, EJB, XML y GIF).

- Admite entornos de desarrollo de aplicaciones basados en GUI

- Se ejecuta en una amplia gama de sistemas operativos, incluidos Windows, LinuxTM, Mac OS X,

5.2.2 FUNCIONAMIENTO DE ECLIPSE

Un Plugin es la unidad más pequeña de la plataforma Eclipse que se puede desarrollar y distribuir por separado. Existe un Kernel conocido como plataforma Runtime toda la funcionalidad de la plataforma Eclipse está realizada con plugins. Los complementos están codificados en Java. El workbench declara un punto de extensión para las preferencias del usuario. Platform Runtime descubre el conjunto de complementos disponibles, lee sus manifiestos y crea un registro enchufable en memoria. A cada plug-in se le asigna su propio cargador de clases Java que es el único responsable de cargar sus clases. *[29]*

Cada complemento declara explícitamente su dependencia de otros complementos de los que espera acceder directamente a las clases, y controla la visibilidad de las clases públicas y las interfaces en sus bibliotecas. El espacio de trabajo consiste en uno o más proyectos de alto nivel, donde cada proyecto se asigna a un directorio especificado por el usuario en el sistema de archivos.

5.3 PHPUnit

PHPUnit es un Fraamework de pruebas unitarias para el lenguaje de programación PHP. Desarrollado por Sebastian Bergmann el 15 de marzo de 2004, esta es una instancia de la arquitectura xUnit, PHPUnit permite probar la lógica de un software escrito en PHP, el código PHP se traduce a nivel de servidor, significa que el código PHP nunca llega al lado del cliente, esta se llega a través de la Interfaz. *[63]*

Cuando se ha identificado, diseñado y creado un plan de prueba correcto, a través de PHPUnit se puede ejecutar de manera inmediata el caso de prueba seleccionado. PHPUnit genera informes detallados de errores, simplifica la integración del Software, valida un valor de retorno a una llamada de método. En la ***Figura 5.2*** se muestra la arquitectura PHPUnit.

PHPUnit 5 requiere PHP 5.6, PHP 7.0, PHP 7.1. PHPUnit también requiere del archivo PHAR que es la extensión que proporciona una manera de colocar aplicaciones PHP enteras dentro de un único fichero llamado "phar" para una distribución e instalación sencillas. *[65]*

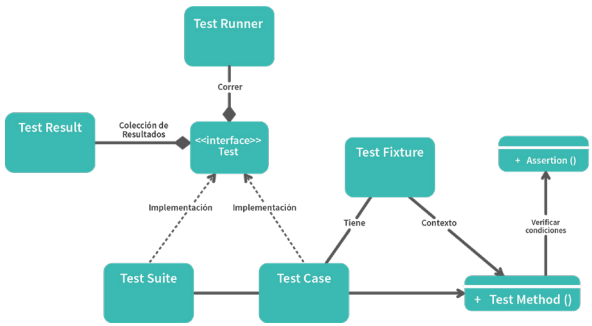


Figura 5. 2 Arquitectura PHPUnit

Existen varios frameworks de desarrollo guiado por pruebas que han llegado a conocerse colectivamente como xUnit. La arquitectura de PHPUnit se basa en la arquitectura de xUnit, esta arquitectura consiste en las clases Caso de Prueba (TestCase), Corredor de Prueba (TestRunner), Instalación de Prueba (TestFixture), Banco de Prueba (TestSuite) y Resultado de Prueba (TestResult).

A continuación, se muestra la descripción de la Arquitectura de PHPUnit: *[36]*

- Caso de Prueba ***(TestCase)*** es la clase base de la prueba unitaria, que contiene métodos de prueba ejecutables.

- Implementa la interfaz Prueba y su método correr ().

- La clase Corredor de Prueba ***(TestRunner)*** es para extender PHPUnit y ejecutar múltiples casos de prueba a la vez y reportarlos.

- La clase Instalación De Prueba ***(TestFixture)*** se utiliza para garantizar el aislamiento de la prueba y la creación de un entorno de prueba separado para cada método de prueba. Habilita proporcionando el caso de prueba con la configuración común y la funcionalidad de desmontaje para el nivel de clase y método. Proporciona un contexto compartido común para los métodos de prueba, donde el entorno se crea desde cero para cada prueba.

- ***Fixture:*** el conjunto de recursos o datos comunes que necesita ejecutar una o más pruebas *[40]*

- El Banco de Prueba ***(TestSuite)*** es una clase creada para agrupar los Casos de Prueba ***(TestCase)*** y también implementa la interfaz Prueba. Hace posible ejecutar múltiples Casos de Prueba ***(TestCase)*** y se puede usar con el corredor de Pruebas ***(TestRunner).***

- La clase Resultado de Prueba ***(TestResult)*** se usa para recopilar los resultados del Banco de Prueba ***(TestSuite)*** y los casos de Prueba ***(TestCase).***

5.3.1 PRUEBAS UNITARIAS EN PHPUnit

A continuación, se muestra cómo se puede escribir pruebas unitarias utilizando PHPUnit ejecutando las operaciones de PHP: *[66]*

- Las pruebas para una clase *Class* van en una clase *ClassTest*
- *ClassTest* hereda de *PHPUnit\Framework\TestCase*
- Las pruebas son métodos públicos que se nombran *test**
- Dentro de los métodos de prueba, los métodos de aserción como *assertSame ()* se utilizan para afirmar que un valor real coincide con un valor esperado.

5.3.2 DEPENDENCIAS ENTRE MÉTODOS EN PHPUnit

Estas dependencias permiten la devolución de una instancia de prueba:

- Un método productor, es un método de prueba que cede su unidad bajo prueba como valor de retorno.
- Un método consumidor, es un método de prueba que depende de uno o más productores y sus valores de retorno.
- *@depends* expresa dependencias entre los métodos de prueba.

5.3.3 ESTÁNDAR DE CODIFICACIÓN PARA PHP

Un estándar de codificación es un conjunto de reglas que se utilizan para escribir archivos de código fuente con el objetivo de lograr estructuras de código mucho más comprensibles e identificables para otros programadores diferentes al autor. [64]

A continuación, se muestra algunas reglas que debe seguir el estándar de codificación antes de realizar las pruebas:

- Descripción de las funcionalidades a través de comentarios en el código.
- Declaración de los tipos de parámetros de funciones ejemplo: ***function sumar ($a /*Integer*/)***
- El código debe comportarse de acuerdo con la especificación, debe utilizar un mínimo de recursos en tiempo y memoria.
- El código debe ser fácil de mantener y depurar

5.4 SELENIUM

Fue originalmente desarrollado por Jason Huggins en 2004 estaba probando una aplicación interna en ThoughtWorks. Siendo un tipo inteligente, se dio cuenta de que había mejores usos de su tiempo que pasar manualmente las mismas pruebas con cada cambio que hizo. Desarrolló una biblioteca de Javascript que podría impulsar las interacciones con la página, lo que le permite volver a ejecutar pruebas automáticamente contra varios navegadores. Esa biblioteca finalmente se convirtió en Selenium Core, que subyace en todas las funciones del Selenium Remote Control(RC) y Selenium IDE *[56]*

Selenium es una herramienta de código abierto para automatizar aplicaciones web, ofrece diferentes funciones adaptadas para realizar pruebas en aplicaciones web. A través de su interfaz permite la comparación de resultados esperados con el comportamiento real de la aplicación. Una de las ventajas que ofrece Selenium es la ejecución de pruebas en diferentes navegadores.

En la *Figura 5.3* se muestra la arquitectura que conforma Selenium

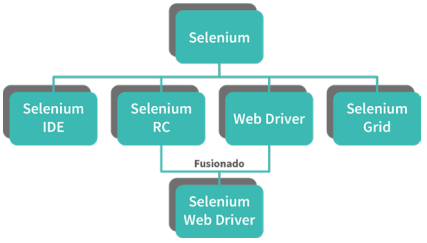


Figura 5. 3 Arquitectura Selenium

5.4.1 SELENIUM IDE

IDE es la herramienta que utiliza para desarrollar casos de prueba, contiene un menú contextual, permite seleccionar un elemento de interfaz y posteriormente se selecciona una lista de comandos de Selenium con parámetros predefinidos de acuerdo con el contexto del elemento UI seleccionado *[60]*

Es el entorno de desarrollo integrado para la construcción de casos de prueba, contiene una función donde se graban las acciones del usuario almacenados en un Script que se ejecuta. Es compatible con diferentes navegadores:

- Google Chrome
- Internet Explorer 7, 8, 9, 10 y 11
- Firefox
- Safari

5.4.2 SELENIUM WEB DRIVER

Selenium-WebDriver se desarrolló para admitir las páginas web dinámicas donde los elementos de una página pueden cambiar sin que la página se vuelva a cargar. Selenium-WebDriver realiza llamadas directas al navegador utilizando el soporte nativo de cada navegador para la automatización. En la *Figura 5.4* se muestra el funcionamiento de WebDriver

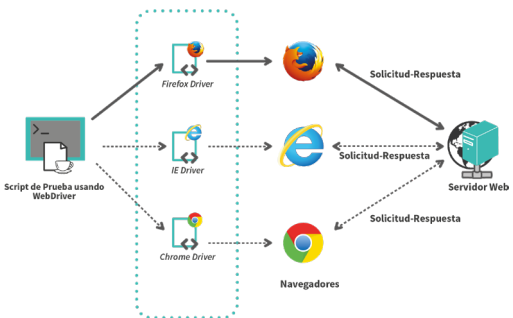


Figura 5. 4 Arquitectura Selenium WebDriver [62]

5.4.3 SELENIUM RC

Selenium Server recibe comandos de Selenium, los interpreta e informa de nuevo a su programa los resultados de ejecutar una prueba.

Selenium es un servidor escrito en Java que acepta comandos al navegador vía HTTP. RC hace posible escribir pruebas automatizadas para aplicaciones web, en cualquier lenguaje de programación lo que permite una mejor integración de Selenium a entornos de prueba existentes. *[60]*

5.4.4 SELENIUM GRID

Selenium Grid permite ejecutar sus pruebas en paralelo, es decir, diferentes pruebas se pueden ejecutar al mismo tiempo en diferentes máquinas remotas.

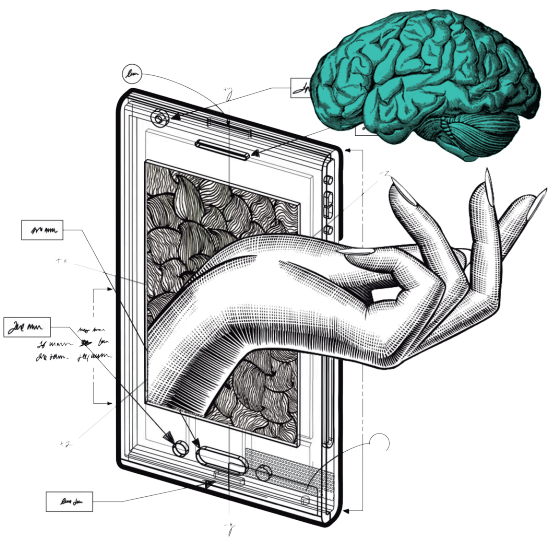
Esto significa dos ventajas importantes:

- Las pruebas se agilizan porque se pueden ejecutar en diferentes maquinas, se reduce el tiempo de ejecución.

- Las pruebas se pueden ejecutar en diferentes entornos de manera paralela, si es específicamente una se puede observar su comportamiento en diferentes contextos.

CAPÍTULO 6

RED SOCIAL-ACADEMICA FCC BUAP



6.1 *Red Social*

6.2 *Tutoría*

6.3 *Ambiente de Trabajo*

6.4 *Tecnología*

6.5 *Descripción General de los Usuarios*

6.6 *Modelización*

RED SOCIAL-ACADEMICA FCC BUAP

La aplicación de las Pruebas de Software para este trabajo de investigación se aplicará para el proyecto de Investigación de la Red Social para estudiantes de la Facultad de Ciencias de la Computación BUAP, en este capítulo se describirá el funcionamiento de esta red Social.

El enfoque y desarrollo de esta red se enfocó hacia los estudiantes de la Facultad de Computación, permitiendo la interacción entre la comunidad estudiantil, además de tener a su disposición herramientas académicas, que incluyen: elaboración de ruta crítica, proyección de cursos, visualización de plan de estudios, etc. Se crea una aplicación web que permitiera la interacción en los diferentes ámbitos que se viven dentro de la faculta, tanto académico como social.

El diseño de la aplicación web se realizó a través del lenguaje de modelización UML, donde se muestran los diferentes componentes de la red social. El desarrollo de la red social-académica que se aplicó con una metodología en todo el periodo del servicio social, esta metodología fue SCRUM, esta metodología ágil permitió llevar a cabo la planeación y ejecución de forma integral de la aplicación.

6.1 RED SOCIAL

La Red Social Académica FCC BUAP brinda la oportunidad a los usuarios de registrarse para tener comunicación entre grupos, tener un perfil donde pueda comunicarse, compartir archivos y publicar contenido académico.

Entre los objetivos a cumplir en esta aplicación es fomentar la comunicación en la FCC haciendo un ambiente colaborativo entre alumnos, catedráticos y miembros administrativos pertenecientes a la comunidad estudiantil.

Algunas actividades que los usuarios realizan dentro de la aplicación son las siguientes:

- *Solicitud de Amistad:* Permite al usuario enviar una invitación a uno o diversos usuarios para la creación de un vínculo de amistad.

- *Mensajes Personales (Inbox):* La mensajería dentro de la red social se puede llevar a cabo entre dos o más miembros de la red.

- *Perfil:* Dentro del perfil el usuario tiene la administración completa de su información personal, como foto, descripción, esté puede añadir nuevas secciones para que los demás usuarios conozcan sus gustos, intereses, etc.

- *Grupos:* La creación de grupos en donde el usuario principalmente tiene la oportunidad de crear un grupo de trabajo, compartir información y contenido multimedia.

- *Avisos:* Contenido relevante dentro de la Red Social

- *Crear Publicación:* Los usuarios pueden crear estructuras de datos a modo de publicación, la cual es publicada en el timeline del usuario

- *Newsfeed:* Es un apartado donde los usuarios pueden ver las publicaciones que los demás o los propios usuarios realizan

6.2 TUTORÍA

Esta modulo tiene por objetivo orientar al alumno respecto a sus materias, visualizando su avance curricular. El alumno visualiza el mapa curricular de la carrera y ver cuáles son los prerrequisitos de cada una de ella, así como el valor en créditos de las materias.

- *Tutoría:* Orienta al alumno sobre los tiempos que requiere invertir para poder terminar la licenciatura, dependiendo del tiempo que desee dedicar para cada periodo.

- *Visualización de mapa curricular:* Se muestra gráficamente el mapa curricular del alumno, donde se visualiza el avance obtenido por periodo. Se visualizan las materias predecesoras y sucesoras correspondientes a cada materia.

- *Datos del Alumno:* Se muestra los datos correspondientes al alumno, nombre, carrera, matricula etc.

- *Visualización Historia Académica:* En este apartado se observa la historia académica del alumno respecto al avance curricular que se obtiene, la calificación obtenida, materias tomadas por periodo.

6.3 AMBIENTE DE TRABAJO

Esta aplicación busca brindar la comunicación entre todos los miembros de la comunidad de la FCC, esta comunicación se realiza a través de canales de información y contenidos multimedia, además de brindar una herramienta de orientación a través de una interfaz a tractiva donde se muestre el avance curricular de cada alumno.

Se muestra las rutas de preferencia respecto a las materias tomadas por el alumno, el espacio físico de trabajo dependerá de los usuarios y el momento en que se utilizará la aplicación. Como primera instancia se observa un plano importante que se encuentra en la BUAP, específicamente en la FCC, tomando en cuenta que los alumnos se encuentran gran parte de su día en las instalaciones antes mencionadas.

6.4 TECNOLOGÍA

A continuación, se describe la tecnología con la que la aplicación web "Red Social Académica FCC BUAP" se desarrolló.

Backend
- **PHP**

Base de datos
- **MySQL**

Frontend
- **HTML5 y CSS3**

Framework
- **Composer**

Con estas tecnologías se permitirá la entrada de caracteres a la aplicación, con la cual se pretende realizar diferentes tipos de tareas que el usuario realizará, las entradas dependerán de la interfaz gráfica en las que se encuentre el usuario. La interfaz gráfica es importante para el usuario ahí se validan las entradas hechas para el usuario, la conexión de un servidor y la conexión de una base de datos con la intención de obtener un resultado, se encuentran la elección de menús, clic sobre elementos, desplazamiento de páginas, validación de formularios, etc.

6.5 DESCRIPCIÓN GENERAL DE LOS USUARIOS

Se realizo el análisis de los diferentes tipos de usuario que tendrían acceso a la Red Social, cabe mencionar que para las pruebas de Software este es un factor importante porque a través de este análisis se determina el acceso a los diferentes módulos que conforman la aplicación, las diferentes tareas que cada usuario debe realizar y los permisos específicos dentro de ella.

Las actividades que se realizaron para determinar el tipo de usuario fueron a través del modelo MODIHC que permite diseñar todos los aspectos involucrados en la interacción entre el humano y una computadora cuando se desarrollan sistemas computarizados describiendo cada una de las características y tipos de usuarios.

- *Usuario Alumno*

Los usuarios con el perfil de Alumno interactúan con el sistema todo el tiempo, realizando tareas específicas como: realización de conversaciones entre miembros de la facultad, creación de grupos y envío de mensajes, avance académico de acuerdo con su mapa.

- *Usuario Académico*

El usuario académico tiene comunicación con sus alumnos incluso con compañeros de trabajo, generando grupos de interés realizando diversas actividades, como: Control académico de alumnos observando el avance curricular de cada uno de ellos.

6.6 MODELIZACIÓN

La metodología utilizada para desarrollar esta aplicación fue SCRUM, se eligió esta metodología ágil por los beneficios que ofrece la planificación y la ejecución de tareas específicas por tiempos.

Es importante recalcar que para obtener una mejor conceptualización se realizaron los diagramas UML para entender y crear un esquema que contenga un criterio de abstracción para cada requerimiento. Con estos diagramas se permite fragmentar el Software de acuerdo con sus funcionalidades.

En la *Figura 6.1* se muestra algunas funcionalidades que el alumno realiza dentro de su perfil en el Software.

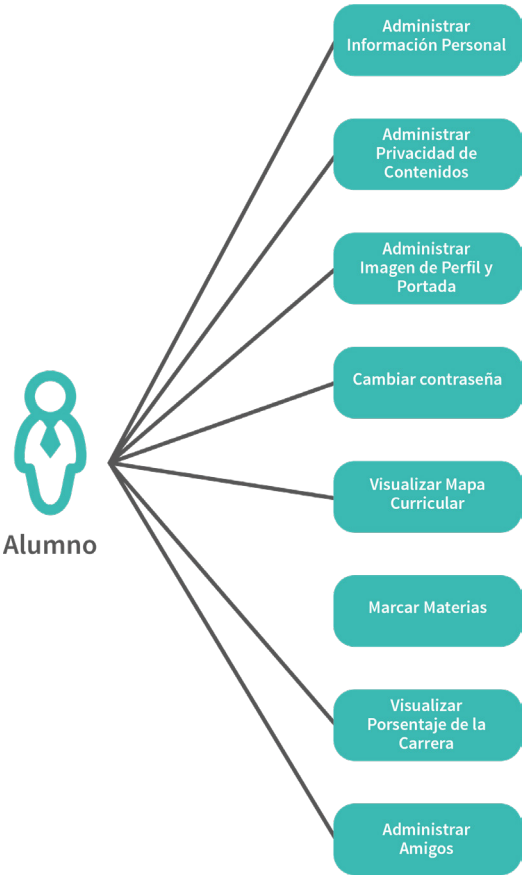


Figura 6. 1 Caso de Uso Personal

Anteriormente mencionamos que para obtener una perspectiva diferente del sistema se realiza la modelización del Software, se observa un caso de uso del Software, y por cada caso de uso se muestra un diagrama de secuencia, que como su nombre lo indica muestra paso a paso el camino que recorre una tarea específica dentro del Software. En la *Figura 6.2* se muestra el diagrama de secuencia para el caso de uso administrar información.

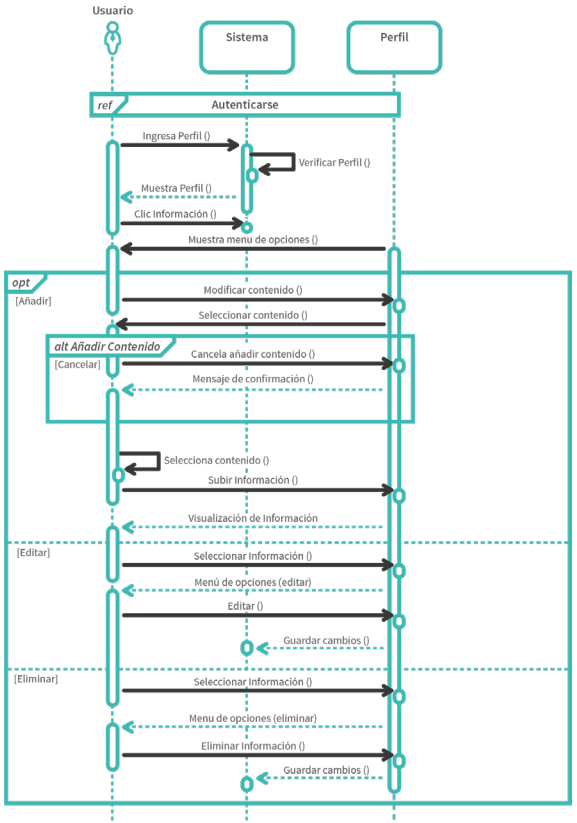


Figura 6. 2 Diagrama de Secuencia Administrar Información

A continuación, se presentan los diagramas de clases correspondientes a la Red Social Académica FCC BUAP, se muestra la estructura de toda la aplicación. En la *Figura 6.3* se muestra el diagrama de clases correspondiente a toda la red Social.

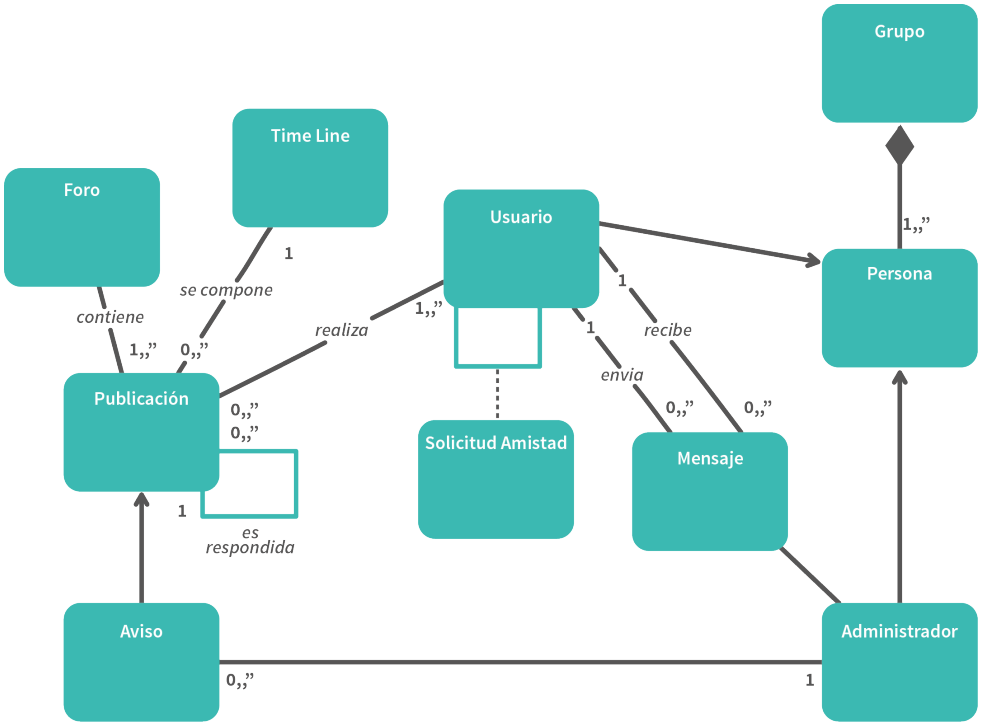


Figura 6. 3 Relación de Clases Red Social

En la *Figura 6.4* se muestra la especificación de cada una de las clases correspondientes a la Red Social.

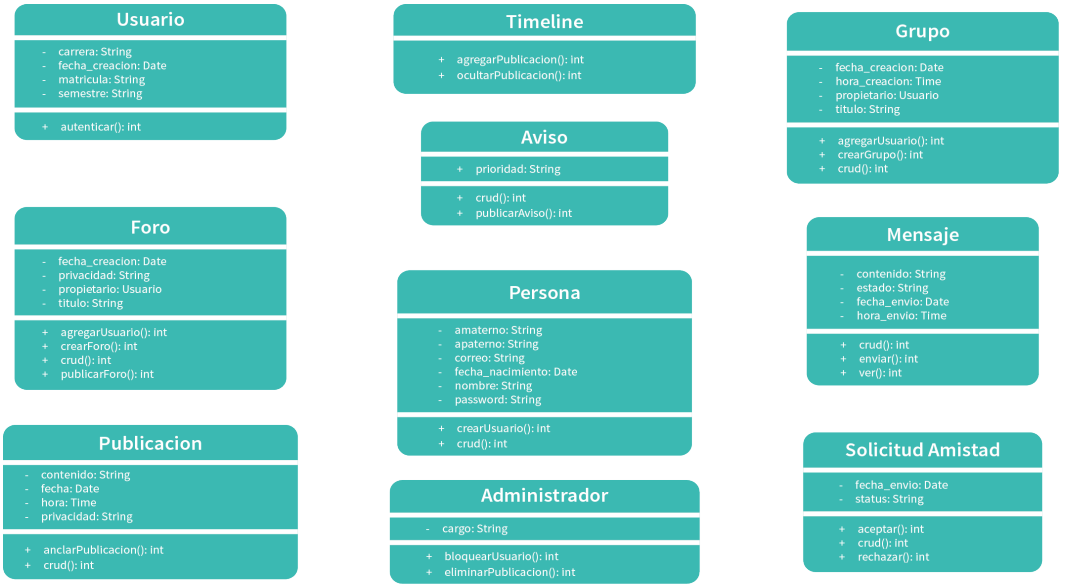


Figura 6. 4 Especificación de Clases Red Social

Figura 6.5 se muestra las clases relacionadas correspondientes a tutoría, con sus respectivos métodos y atributos.

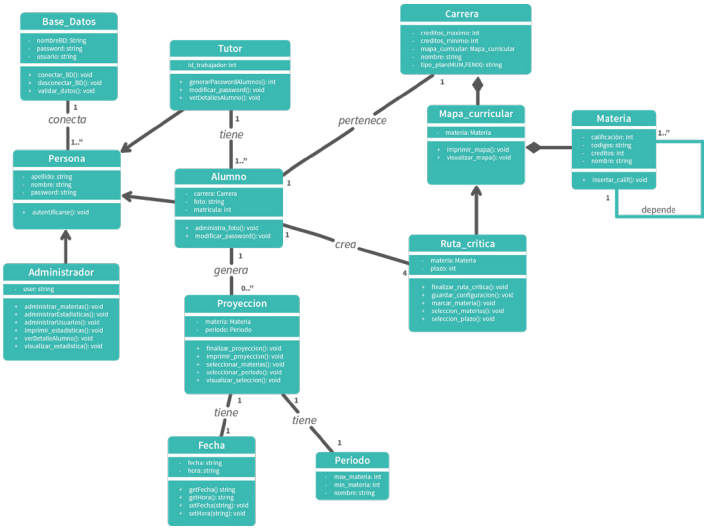


Figura 6. 5 Diagrama de clases Tutoría

El diagrama de navegación permite visualizar los enlaces entre los elementos de la aplicación, en la *Figura 6.6* se muestra el diagrama de navegación de la aplicación.

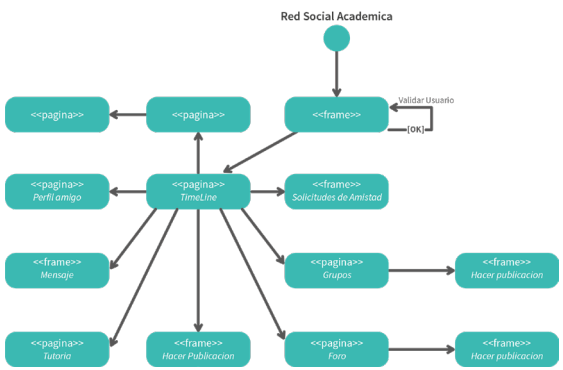


Figura 6. 6 Diagrama de Navegación de la Red Social Académica

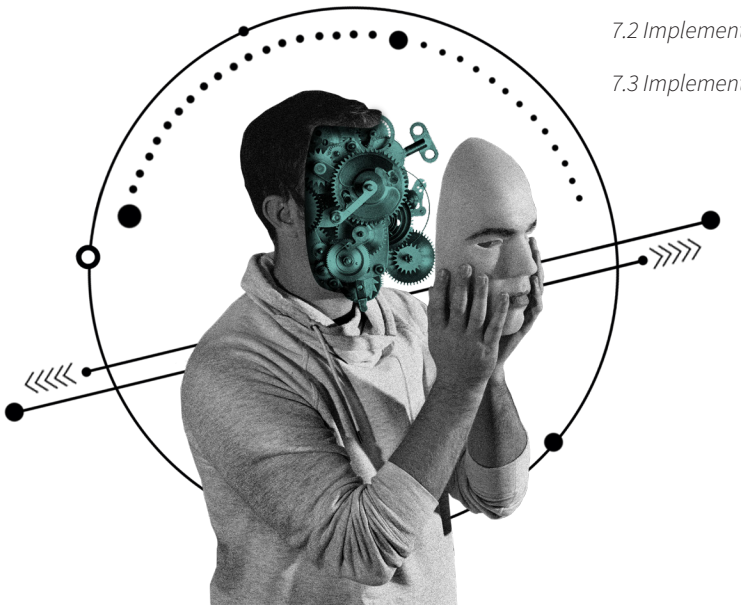
CAPÍTULO 7

IMPLEMENTACIÓN Y RESULTADOS DE LAS PRUEBAS DE SOFTWARE

7.1 Configuración

7.2 Implementación de Pruebas con Selenium

7.3 Implementación de Pruebas con PHPUnit



IMPLEMENTACIÓN Y RESULTADOS DE LAS PRUEBAS DE SOFTWARE

La implementación de las pruebas de Software es muy importante muestra el estado en el que se encuentra el Software, este es el punto exacto para determinar el grado de calidad con la que cuenta el desarrollo del Software, además se muestra la cobertura de requisitos y la funcionalidad del Software.

La generación de datos de prueba fue a través de generación de datos aleatorios. Esta técnica es la más utilizada en la generación de datos de prueba, consiste en generar datos de entrada al azar, estos datos son ingresados para observar el comportamiento del Software.

Las pruebas aleatorias se crearon de manera independiente, debido a que las pruebas que se implementaron fueron pruebas unitarias y para cada módulo se necesitan diferentes datos. Los datos de entrada son seleccionados, implementados, ejecutados y observados. Una vez ejecutados los datos de prueba dentro del módulo se obtienen resultados del comportamiento y de la funcionalidad del módulo. Se realiza la comparación de los resultados esperados con los resultados obtenidos.

El implementar pruebas aleatorias permitió observar el comportamiento de cada módulo, este comportamiento dependía del dato de entrada revelando comportamiento de éxito o por el contrario se presentan errores bajo diferentes circunstancias.

En las siguientes secciones se presenta los resultados obtenidos en la implementación de pruebas de software la estrategia utilizada fue a través de pruebas automáticas, se observó las pruebas con resultados de éxito y de fracaso en casos de prueba para determinar el grado de calidad que la aplicación brinda.

7.1 CONFIGURACIÓN

Para poder implementar las pruebas de Software se utilizó el IDE de Eclipse para las pruebas unitarias de Interfaz se instaló un plugin de Selenium en el IDE de eclipse y posteriormente para realizar las pruebas unitarias en el código se realizaron configuraciones en el sistema y se instaló un plugin de PHPUnit en el IDE de Eclipse, las configuraciones y los plugin permitieron correr las pruebas automáticas para el sistema.

Pruebas de Interfaz:

- Elipse IDE Java Oxygen
- Navegador Google Chrome
- Plugin Selenium chromedriver.exe
- Librerías selenium-java-3.9.1

Prueba de Código:

- Eclipse IDE Java Mars
- PHPUnit
- Navegador Google Chrome
- PDT de PHP development tools
- Plugin para pruebas PHPUnit Support
- Librerías go-pear.phar

7.2 IMPLEMENTACIÓN DE PRUEBAS CON SELENIUM

Con el framework Selenium se implementaron casos de prueba para observar la funcionalidad en las Interfaces. El fin de estas implementaciones consistió en encontrar errores en la red social académica FCC BUAP a través de su interfaz. Las pruebas se realizaron a través de los elementos que conforma esta aplicación.

Cabe mencionar que para implementar este tipo de pruebas se debe conocer los requisitos del Software y el comportamiento esperado para cada módulo. La especificación de requerimientos ayuda a realizar el análisis de que tan apegada esta la aplicación con lo que requiere el cliente.

La implementación para el módulo tutoría se concluyó de manera rápida esto se debe a que cada uno de los elementos probados contaba con un ID, a través de este ID se hace referencia a los elementos para probarlos. Recordemos que en el *Capítulo V* se mencionan las buenas prácticas de programación.

Una buena práctica es incluir el ID para cada elemento, esta práctica disminuyo el tiempo para diseñar la prueba, los elementos de prueba eran fáciles de encontrar. Además, recordemos que para ser más objetivos en probar cualquier Software o aplicación el tester debe ser externo al desarrollo, los ID ayudan al tester a identificar los elementos, se ocupan menos recursos. En la ***Figura 7.1*** se muestra un ejemplo de la asignación de ID's.

```
    <input type="text" name="matricula" id="matricula" />
</div>

<div>
    <label>Contraseña</label>
    <input type="password" name="contrasena" id="contrasena" />
</div>
<div>
    <div>
</div>
```

Figura 7. 1 Asignación de ID´s en el código fuente

El siguiente caso de prueba se enfoca a probar el comportamiento del inicio de sesión para el módulo tutoría, en el código se observa que se busca el elemento que se probará, se hace referencia al elemento a través de su ID y posteriormente se ingresa el dato generado para probar el módulo, en otro caso se implementa la acción que debe seguir el elemento. Ejemplo ***click ().***

```
driver.findElement(By.id("matricula")).sendKeys("201240391");
pausa(500);
driver.findElement(By.id("contrasena")).sendKeys("1234");
pausa(500);
driver.findElement(By.id("submit")).click();
pausa(1000);
```

Figura 7. 2 Ejecución del caso de prueba inicio de sesión del modulo tutoría

El diseño para mostrar los casos de prueba se incluyó en tablas donde se presentan las precondiciones entre cada módulo, algunos valores de entrada que fueron válidos y valores inválidos para observar el comportamiento del Software, las

letras R1, R2 Y R3 muestran la validez de los casos que son los casos válidos, inválidos y la respuesta de ambos, por último, se muestra la observación de los casos.

Precondición:		El usuario tiene que estar de alta dentro del sistema universitario						
Núm Caso	Elementos	Valores Válidos	Respuesta	Valores Inválidos	Respuesta	r1	r2	r3
1	*Matrícula* *Contraseña* *Click(): Iniciar Secion*	*open=/ITISOLbuap* *201240391* *12345* *id= Submit*	*Se inicia sesión correctamente*	*open=/ITISOLbuap* *2012403* *1234* *id= Submit*	*Muestra mensajes de error*	*x*	*x*	*x*
Observación:		*La base de datos valida correctamente los datos, cuando el caso de prueba es inválido muestra mensaje de error que ayuda al usuario a verificar sus datos.*						

Tabla 7. 1 Caso de Prueba iniciar sesión tutoría

A través de la introducción de Selenium y el IDE de Java se pueden observar las pruebas automáticas introduciendo datos predeterminados como se mostró en el ***Caso 1*** de prueba. Existe otra herramienta que se utiliza a través de un complemento de Selenium que se introduce en el navegador, para estas pruebas se utilizó el navegador de Google Chrome.

A través de este complemento se graban los eventos paso a paso que se activan para realizar una transacción en específico para este caso el evento es el ***Click ()*** en un botón específico y la respuesta es desplegar un menú correspondiente al botón. En la ***Figura 7.3*** se muestra la ejecución del caso de prueba a través del complemento de Selenium en el navegador Google Chrome.

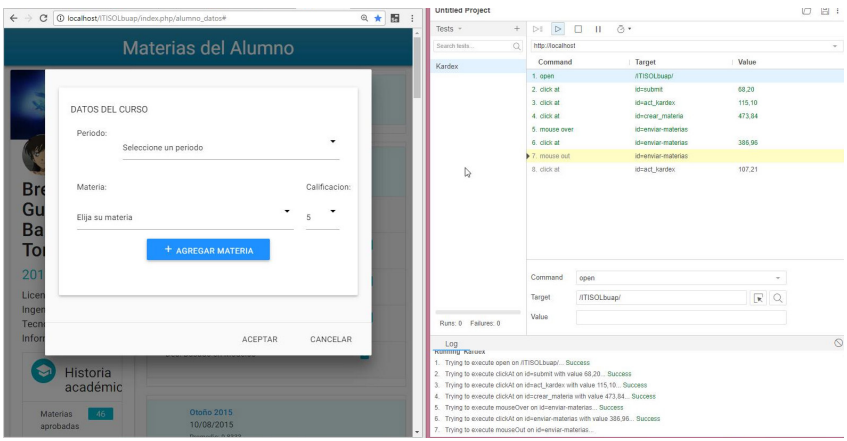


Figura 7. 3 Vista de ejecución Click () Actualizar Kardex

Se observa que los eventos se van ejecutando paso por paso, la referencia a los elementos se hace a través de los ID's y los comandos correspondientes. Cada transacción se va analizando y se muestra el resultado línea por línea. Cuando el análisis termina se comparan todos los resultados de los comandos mostrando un resultado final en la ***Figura 7.4*** se muestra este resultado.

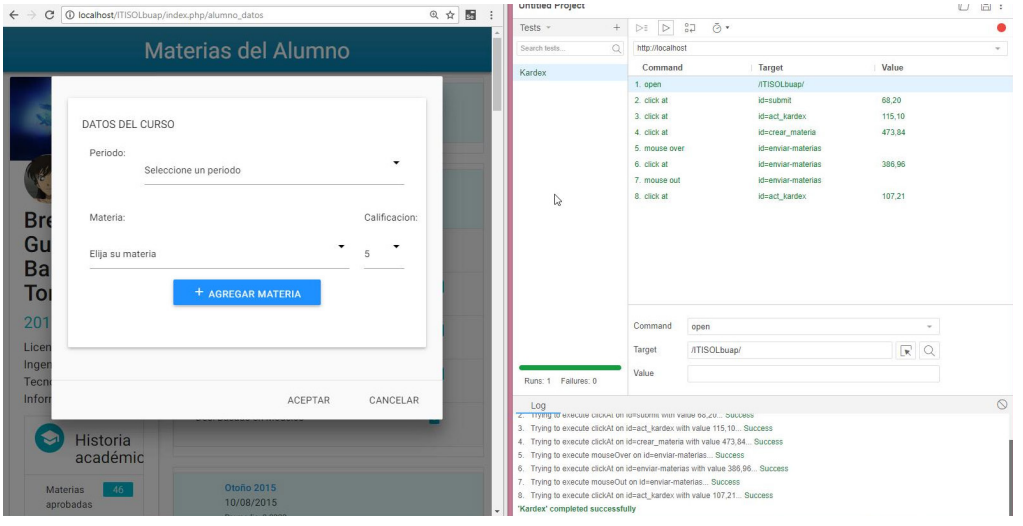


Figura 7. 4 Resultado final de un caso de prueba exitoso

En la *Tabla 7.2* se muestra el resumen del caso de prueba para ejecutar el caso de prueba Actualizar Kardex.

Precondición:		**Se inicio sesión dentro del módulo tutoría**		
Núm Caso	**Elementos**	**Valores Válidos**	**Respuesta**	**r1**
1	*Click (): Actualizar Kardex*	*id= Submit* *id=act_kardex*	*Desplegar formulario*	*X*
Observación:		*El comando respondió correctamente a la solicitud*		

Tabla 7. 2 Caso de prueba Actualizar Kardex

Este caso de prueba evalúa la visualización del Mapa Curricular.

Precondición:		Se inicio sesión dentro del modulo tutoria						
Núm Caso	Elementos	Valores Válidos	Respuesta	Valores Inválidos	Respuesta	r1	r2	r3
3	*Click (): Ver-mapa*	*id= Submit* *css= ver-mapa*	*Muestra Mapa Gráfico*	*id= cerrar-mapa* *id= fault*	*Se genera un bug no cierra el Mapa Gráfico*	*x*		
Observación:		*El boton cerrar no respondio ya que no se encuentra dentro del area donde el usuario*						

Tabla 7. 3 Caso de prueba ver Mapa Gráfico

```
driver.findElement(By.id("ver-mapa")).click();
pausa(2000);
driver.findElement(By.id("cerrar-mapa")).click();
```

```
at org.openqa.selenium.remote.HttpCommandExecutor.execute(HttpCommandExecutor.java:160)
at org.openqa.selenium.remote.service.DriverCommandExecutor.execute(DriverCommandExecutor.java:83)
at org.openqa.selenium.remote.RemoteWebDriver.execute(RemoteWebDriver.java:601)
at org.openqa.selenium.remote.RemoteWebElement.execute(RemoteWebElement.java:279)
at org.openqa.selenium.remote.RemoteWebElement.click(RemoteWebElement.java:83)
at Pruebas.main(Pruebas.java:44)
```

Figura 7. 5 Ejecución del caso de prueba ver mapa gráfico

En la ***Figura 7.6*** Se muestra el Mapa Grafico, se observa que el botón cerrar no está a la vista del usuario.

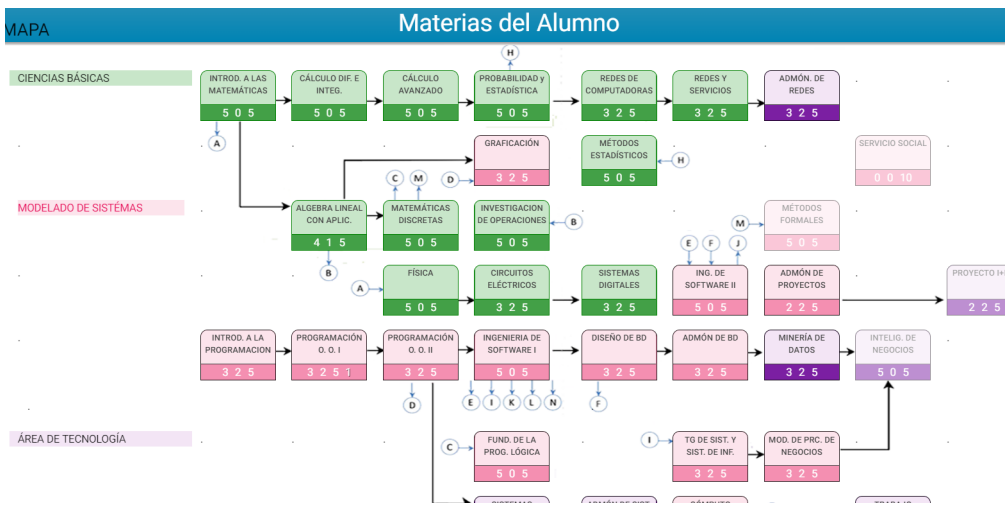


Figura 7. 6 Interfaz del Mapa Gráfico

A continuación, se muestra casos de prueba que pertenecen a los módulos correspondientes a la Red Social, recordemos que el conjunto de esta aplicación se realizó por módulos bajo la metodología SCRUM, cabe mencionar que para los módulos de la red social no se determinaron ID´s para los elementos, esto implicó más tiempo invertido para probar las interfaces de esta aplicación.

La falta de ID's implicó el uso de otras herramientas para la interfaz, inicialmente se inició la búsqueda de ID´s en el código para poder realizar los casos de prueba; el primer inconveniente es que los elementos no contaban con ellos, esto implicó que para realizar los casos de prueba se hizo uso de las herramientas de desarrollador del navegador web Google Chrome para identificar el nombre de los elementos e identificarlos en cada caso de prueba para realizar su prueba automática. En la *Figura 7.7* se observa el uso de la herramienta para identificar los nombres de los elementos

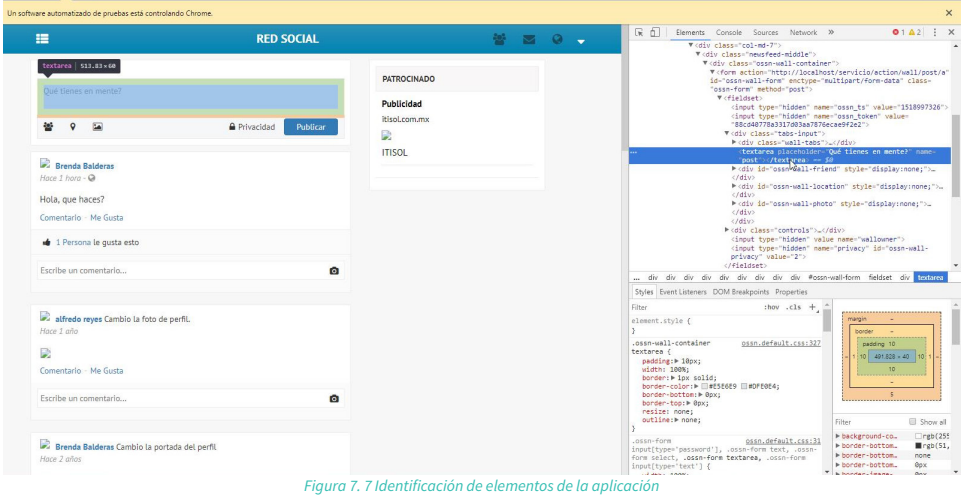


Figura 7. 7 Identificación de elementos de la aplicación

En la ***Figura 7.8,*** se puede observar que los elementos no contienen ID's para identificarlos en las pruebas de Software.

```
<div>
    <input type="text" name="firstname" placeholder="<?php echo ossn_print('first:name'); ?>"/>
    <input type="text" name="lastname" placeholder="<?php echo ossn_print('last:name'); ?>"/>
</div>

<div>
    <input type="text" name="email" placeholder="<?php echo ossn_print('email'); ?>"/>
    <input name="email_re" type="text" placeholder="<?php echo ossn_print('email:again'); ?>"/>
</div>
```

Figura 7. 8 Código fuente sin asignación de ID's

El que unas partes de desarrollo no tengan ID's y otros si los contengan, significa que los desarrolladores en un inicio no decidieron elegir un parámetro de buenas prácticas para estandarizar su programación.

En la ***Tabla 7.4*** se observa el caso de prueba Inicio de Sesión para ingresar a la red Social.

Precondición:		El usuario tiene que estar registrado dentro de la red social						
Núm Caso	Elementos	Valores Válidos	Respuesta	Valores Inválidos	Respuesta	r1	r2	r3
4	*/servicio/ name=password css=a.btn.btn-primary css=input.btn.btn-primary*	*open=/servicio/login username=brenda12345 click() password=brenda12345 click()*	*Inico de sesión*	*open=/ITISOLbuap 2012403 1234 id= Submit*	*Muestra mensajes de error*	*x*	*x*	*x*
Observación:		*La base de datos valida correctamente los datos, cuando el caso de prueba es inválido muestra mensajes de error que ayuda al usuario a verificar sus datos.*						

Tabla 7. 4 Caso de prueba Inicio de Sesión Red Social

A continuación, se muestra los mensajes obtenidos cuando se ingresan datos de prueba inválidos, los mensajes ayudan en este caso a guiar al usuario a obtener información de en qué punto de la interfaz no ingreso datos correctamente.

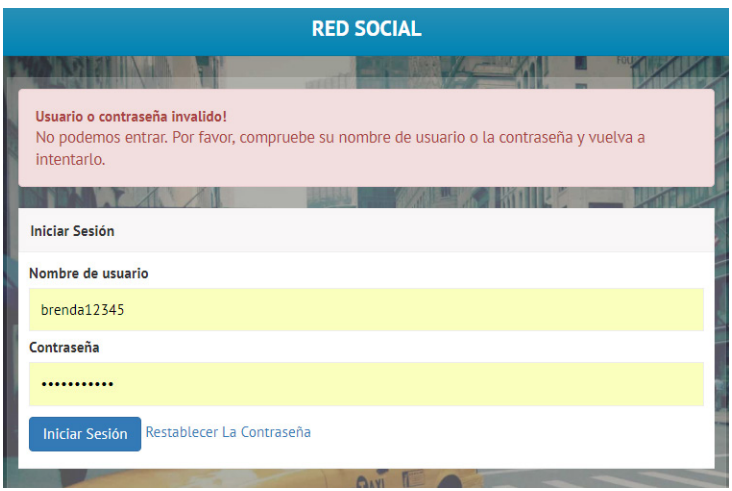


Figura 7. 9 Mensajes de error

Ahora se muestra el caso de prueba para publicar un post en la red Social. En la ***Figura 7.10*** se muestra el código de ejecución para este caso de prueba.

```
driver.findElement(By.name("post")).sendKeys("Esto es una simple prueba.");
pausa(500);
driver.findElement(By.className("ossn-wall-post")).click();
pausa(500);
```

Figura 7. 10 Ejecución para realizar un post en la red social

Precondición:		Se inicio sesión de la red social, el usuario se encuentra en la página principal		
Núm Caso	Elementos	Valores Válidos	Respuesta	r1
5	*/servicio/home* *name=post* *input='publicar'*	*open=/servicio/home* *post=Esto es una simple prueba* *input@value='Publicar'* *oss-vall-post=click()*	*Inserta publicación*	*X*
Observación:		*El botón publicar realiza la función correctamente, en un caso contrario si se desea publicar sin realizar un post se observa un mensaje de verificación.*		

Tabla 7. 5 Caso de prueba Insertar publicación

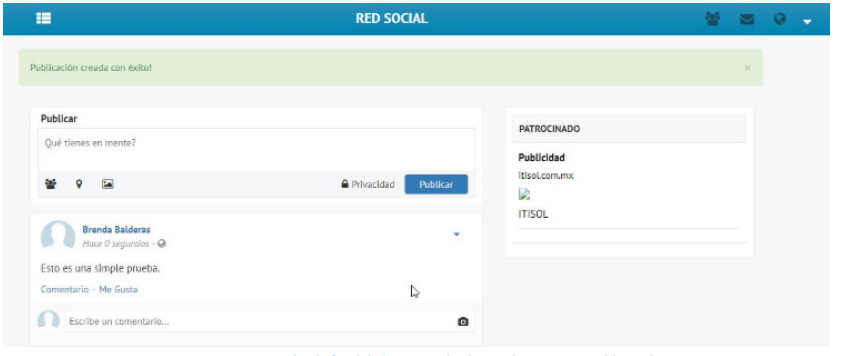


Figura 7. 11 Resultado final de la ejecución de prueba Insertar publicación

En la *Tabla 7.6* se muestra la confirmación de amistad dentro de la red social. Cabe mencionar que se necesita la solicitud de amistad previa, la confirmación se registra en la base de datos agregando un nuevo amigo.

Precondición:		Se inicio sesión de la red social, el usuario se encuentra en la página principal		
Núm Caso	Elementos	Valores Válidos	Respuesta	r1
6	*/servicio/home oss-notifications-friends= notificacion input=' aceptar '*	*open=/servicio/home oss-notifications-friends #add-friends-4>.btn-primary*	*Acepta solicitud de amistad*	*x*
Observación:		*La confirmación se registra en la base de datos agregando un nuevo amigo. En la interfaz se puede observar el nombre de la persona que ha enviado la solicitud.*		

Tabla 7. 6 Caso de prueba Solicitud de amistad.

En la *Figura 7.11* se observa el resultado de caso de prueba Solicitud de amistad, se observa cuando se obtiene una solicitud y se acepta.

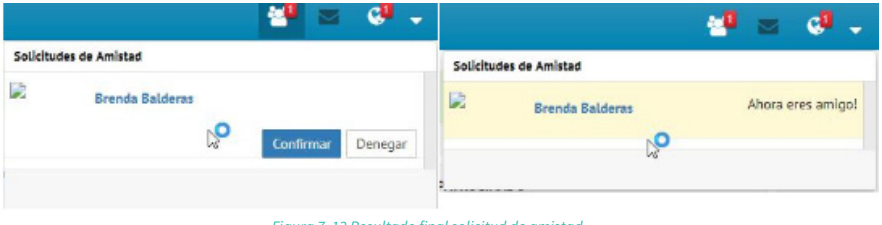


Figura 7. 12 Resultado final solicitud de amistad.

Como caso contrario se observa la eliminación de un amigo, se observó el comportamiento de esta tarea, en la ***Tabla 7.7*** se observa el caso de prueba aquí se observó sólo un pequeño error en los menajes de confirmación.

Precondición:		**El usuario está dentro de la aplicación.**		
Núm Caso	**Elementos**	**Valores Válidos**	**Respuesta**	**r1**
7	*/servicio/home css=#sidebar-toggle>a //a[contains(text(),'Amigos')]*	*click()=btn-danger*	*Acepta solicitud de amistad*	*x*
Observación:		*El mensaje de confirmación no es muy intuitivo, el mensaje se deberá cambiar.*		

Tabla 7. 7 Caso de prueba eliminar un amigo.

Para este caso de prueba se obtuvo buenos resultados, en cuanto a las transacciones y tareas a realizar, se observó un pequeño defecto en él envió de mensaje, porque se observa que se ha eliminado una solicitud de amistad cuando en realidad se está eliminando un amigo. Los mensajes también deben de cumplir la especificación de ser intuitivos. En la ***Figura 7.13*** se observa el resultado para eliminar un amigo.

Figura 7. 13 Resultado final eliminar un amigo

Una de las partes importantes diseñadas para la Red Social es la comunicación entre sus miembros a continuación, se muestra el caso de prueba para enviar y recibir mensajes entre los miembros de la Red Social. Posteriormente se observa el caso de prueba grupos.

En la *Tabla 7.8* se muestra el caso de prueba enviar mensaje.

Precondición:		El usuario está dentro de la aplicación. Se elije en específico a un amigo que será el receptor del mensaje.		
Núm Caso	Elementos	Valores Válidos	Respuesta	r1
8	*/servicio/home* *className=userlink* *id=profile-message*	*name=message* *click()=btn-primary*	*SendKeys= "Esto es un mensaje de prueba"*	*X*
Observación:		*El metodo SendKeys funciona correctamente ya que la transacción de envió del mensaje es correcta.* *Se envía el mensaje a su receptor.*		

Tabla 7. 8 Enviar Mensaje

En la ***Figura 7.14*** se observa el resultado final para este caso de prueba. La interfaz probada hasta el momento contiene colores, atributos y componentes que son intuitivos, estas prueban demuestran que la red social es bastante intuitiva para los usuarios y que cumple claramente con los requisitos mencionados en el capítulo anterior.

Figura 7. 14 Resultado final caso de prueba enviar mensaje

En la ***Figura 7.15*** se muestra el código para generar la prueba automática utilizada para verificar agregar un grupo dentro de la red social.

```
driver.findElement(By.cssSelector("#sidebar-toggle>a")).click();
pausa(500);
driver.findElement(By.cssSelector("#menu-content>li:last-of-type")
pausa(500);
driver.findElement(By.id("ossn-group-add")).click();
pausa(2000);
driver.findElement(By.name("groupname")).sendKeys("Grupo Prueba");
pausa(500);
driver.findElement(By.className("btn-primary")).click();
pausa(1000);
```

Figura 7. 15 Código Fuente para la prueba automática agregar un grupo

Precondición:		El usuario está dentro de la aplicación.		
Núm Caso	Elementos	Valores Válidos	Respuesta	r1
9	cssSelector=sidebar-toggle>a cssSelector=menu-content>li:last-of-type id=ossn-group-add	name=groupname sendkeys=Grupo Prueba click()=btn-primary	SendKeys= "Grupo creado con exito"	X
Observación:		Se crea un nuevo grupo con el nombre anteriormente dado.		

Tabla 7. 9 Caso de prueba crear un nuevo grupo

Figura 7. 16 Resultado final caso de prueba crear grupoa

En casos de prueba anteriores se observó la importancia de una buena práctica de programación, los ID´s ayudaron a realizar las pruebas rápidas y de manera eficiente, con los ID´s se identificaron los elementos necesarios para cada prueba.

El apartado de búsqueda se encuentra en el menú principal su manera de presentación no es intuitiva por lo que confunde a sus usuarios respecto a sus funciones. En la ***Figura 7.17*** se muestra el menú principal de la red-social.

Figura 7. 17 Menú principal Red-Social

En la Tabla 7.10 se muestra el caso de prueba búsqueda.

Precondición:		**El usuario está dentro de la aplicación. Requiere realizar una búsqueda de un miembro en especial.**		
Núm Caso	**Elementos**	**Valores Válidos**	**Respuesta**	**r1**
10	*cssSelector=sidebar-toggle>a click() cssSelector=menu-content>li:last-of-type*	*name=q sendkeys=enrique cssSelector=.oss-form.oss-search*	*SendKeys=name*	*X*
Observación:		*Se muestra los datos del nombre buscado.*		

Tabla 7. 10 Caso de prueba búsqueda

En cuestión de intuición de interfaz se observó que el menú principal se encuentra escondido por lo que no es intuitivo y es difícil de utilizar para los usuarios. Este menú no indica realmente lo que hace, además de que no está a la vista del usuario.

7.3 IMPLEMENTACIÓN DE PRUEBAS CON PHPUnit

En este apartado se realizaron pruebas unitarias bajo el código fuente, los casos de prueba se aplicaron con el framework PHPUnit. Los casos de prueba se implementaron realizando métodos de prueba Productores esto quiere decir que son métodos de prueba que ofrecen a la parte del Software un valor de retorno.

Al ser un framework de pruebas unitarias PHPUnit soporta la declaración de dependencias explicitas entre los métodos de prueba, permite que los métodos no tengan un orden específico.

Para implementar las pruebas unitarias en PHP se utilizaron diferentes métodos:

- *fixture ():* es un estado inicial de la aplicación cuando se ejecuta una prueba.
- *setUp ():* este método se utiliza para inicializar variables, abrir la conexión de archivos, etc., preparando el entorno para la prueba.
- *test*:* es el nombre de las pruebas y son métodos públicos.

En la *Figura 7.18* se observa la implementación del caso de prueba que verifica si la matrícula del alumno corresponde a una carrera en específico.

```php
<?php
require('CarreraModel.php');
class TestCarreraModel extends PHPUnit_Framework_TestCase
{
    protected $fixture;

    protected function setUp()
    {
        $this->fixture = new Carrera_Model();
    }

    public function testDatosCarreraAlumno()
    {
        $datosCarrera = $this->fixture->datos_carrera_alumno('201240391');

        $this->assertEquals('Ingenieria en Tecnologias de la Informacion',
                $datosCarrera['nombre']);
    }
```

Figura 7. 18 Código fuente verificación de carrera

Se muestra que este caso de prueba obtuvo un resultado exitoso. Observemos que se muestra el resultado, los elementos que fueron analizados y el tiempo de ejecución para el caso de prueba.

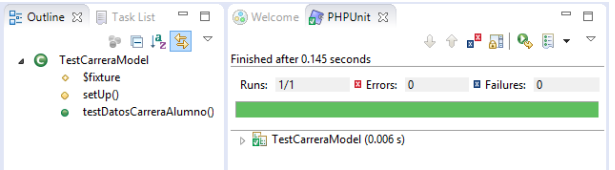


Figura 7. 19 Resultado del caso de prueba de la carrera

A continuación, se muestra un caso de prueba donde el resultado concluyo en error, como mencionamos anteriormente son casos de prueba específicamente productores donde se regresan valores de retorno, cuando el resultado es erróneo en el caso de prueba quiere decir que la comparación o el valor esperado no es el correcto.

```php
public function testNumeroReprobadas()
{
    $reprobadas = $this->fixture->numero_reprobadas('201240391');

    $this->assertEquals(2, $reprobadas);
}
```

Figura 7. 20 Caso de prueba verificación de materias

En la *Figura 7.21* se muestra el resultado final del caso de prueba.

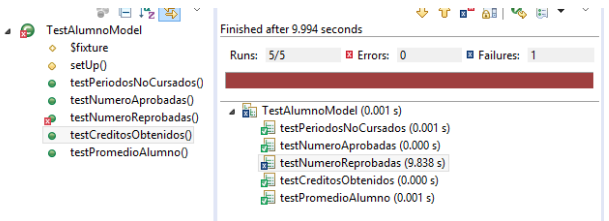
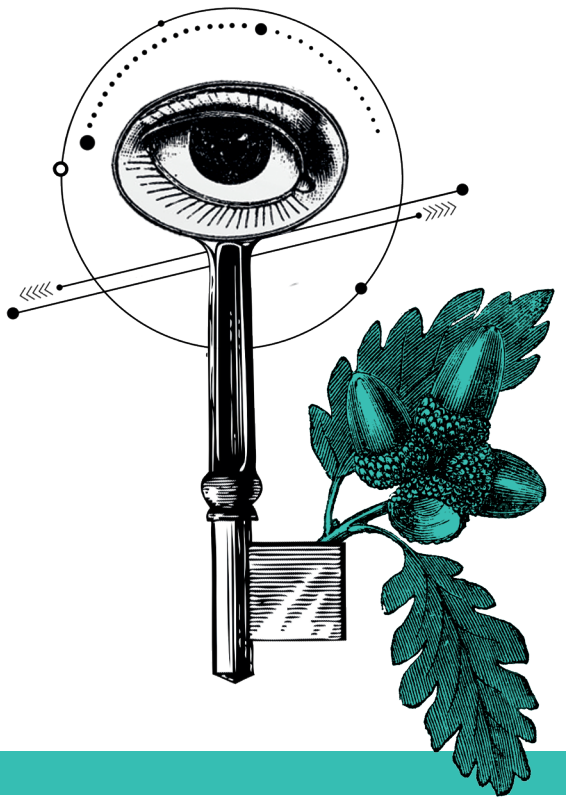


Figura 7. 21 Resultado caso de prueba verificación materias

CAPÍTULO 8

CONCLUSIONES Y TRABAJO FUTURO



CONCLUSIONES Y TRABAJO FUTURO

Como se observó en este trabajo de investigación, adentrarse a las pruebas de Software es un tema extenso pero que a la vez ofrece grandes beneficios, ofrece la oportunidad de realizar procesos a través de estrategias para obtener un Software de calidad, un Software que tiene la oportunidad de ser exitoso porque es revisado, analizado y corregido. Las pruebas de Software ayudan a obtener con gran claridad el funcionamiento de un Software o aplicación, ayuda a entender las necesidades del cliente.

Este trabajo de investigación pretende generar el interés por las pruebas de Software además de enfatizar la importancia que tiene generar procesos de creación de Software con parámetros, métricas, estrategias y factores de calidad.

Implementar pruebas de Software identifica perfectamente las deficiencias y/o funcionalidades del Software, brinda un panorama exacto de los posibles errores que se pueden generar en un futuro, las pruebas contribuyen a conocer toda la estructura del Software y las tareas específicas que un usuario realizara bajo un determinado contexto.

Este trabajo de investigación se enfocó a implementar pruebas unitarias con dos Frameworks. El Framework Selenium se enfatizó a realizar pruebas unitarias respecto a la interfaz de la Red Social Académica FCC BUAP, con PHPUnit se implementaron pruebas unitarias respecto a la lógica de la Red Social. Estos frameworks ayudaron a implementar pruebas de Software unitarias de manera automática con el fin de eliminar errores que una prueba de manera manual puede generar.

Al implementar las pruebas unitarias con estos frameworks se dio la pauta para observar que las herramientas no son tan intuitivas, requieren de una serie de pasos para ponerlas en marcha, configuraciones para que sean compatibles en diferentes plataformas a utilizar, además del tiempo de aprendizaje para entenderlas.

Inculcar un proceso de creación de Softwares con calidad es de suma importancia, porque significa hacer bien las cosas desde sus inicios y como conclusión se obtiene un buen trabajo.

La búsqueda de un framework donde se observe casos de prueba, además de unitarios, es un punto importante para el trabajo futuro, además de implementar otros niveles de prueba no sólo pruebas unitarias, sino también de integración, regresión etc. Combinando otras estrategias de generación de datos no sólo aleatorios, además de implementar estrategias de la caja negra y caja blanca, para obtener una cobertura total de la aplicación.

Ofrecer la calidad de un Software significa estar en la búsqueda continua de mejorar un Software a través de sus procesos de creación, cada componente del Software debe ser revisado para posteriormente ser integrado con la finalidad de obtener un Software eficiente y funcional. Parte importante de esta creación es el trabajo realizado por los miembros del equipo. La estructura organizacional es parte fundamental en la creación del Software es aquí donde depende los costos, el tiempo, la funcionalidad del Software y el cumplimiento de requisitos resumidos en palabras es la calidad Software.

El análisis en la mejora continua en los procesos de Software se observaría a través de la administración correcta de recursos, además de ofrecer las ventajas competitivas y los retos personales de cada equipo, se crearían registro de tiempos, actividades y planes de contingencia cuando algo sale mal, esto se lograría con la implementación correcta de una metodología.

Este trabajo de tesis da pie a la investigación respecto al impacto que tiene implementar CMMI *(Capability Maturity Model Integration)* en conjunto con el PSP *(proceso personal de software)* donde se analizarían estrategias para las buenas prácticas del desarrollador influyendo sus habilidades y los cambios que se pueden realizar dentro de la organización para que los procesos mejoren y la produ

En la búsqueda de la perfección en los procesos de creación de Software se implementaría el modelo de CMMI abordando los niveles de madurez del Software y la generación de cambios en el mejoramiento de la organización además de la implementación del análisis respecto al impacto que se tiene implementando el PSP *(Personal Software Process) (Humphrey,1995)*, analizando las fortalezas de cada miembro del equipo como base del mejoramiento y pensar en las mejores prácticas o los pasos a seguir para implementarlo

"Sé un punto de referencia de calidad. Algunas personas no están acostumbradas a un ambiente donde la excelencia es aceptada."

– Steve Jobs

BIBLIOGRAFÍA

[1] González, L. (2012). ***Introducción al Software Testing***. Validación y Pruebas, 35. [4]

[2] IEEE Std 610.12-1990. (1990). ***IEEE Standard Glossary of Software Engineering Terminology.*** Standards Coordinating Committe of the Computer Society of the IEEE, 84. [4,27,74]

[3] ***Modelo McCall*** [Ilustración]. Recuperado de Fuente: http://campusvirtual.uned.ac.cr/lms/pluginfile.php/530311/mod_resource/content/2/Capitulo_6_-_Calidad_del_software_v1.pdf

[4] Campaña, M. (2014, mayo). ***Modelo Boehm*** [Ilustración]. Recuperado de https://es.slideshare.net/franzmarulanda3/material-monster-is-ii-emco

[5] Garimella, N., Krishna, P., & Deva, S. (2013). ***Software Testing Techniques and Documentation.*** International Journal of Application or Innovation in Engineering & Management (IJAIEM), 2(3), 5. [3]

[6] Kaner, C. (2003). ***What Is a Good Test Case?*** Improving the Education of Software Testers, 16. doi: EIA-0113539 ITR/SY+PE

[7] Bingue, E. W.P, & Cook, D. A. (s.f.). ***A Practical Approach to Verification and Validation.*** [12]

[8] Wallace D. R. and R. U. Fujii. ***Software verification and validation: an overview,*** IEEE Software. [10]

[9] Collofello, J. S. (1988). ***Introduction to Software Verification and Validation (Ed. rev.).*** Tempe, Arizona: Carnegie Mellon University. [18]

[10] Farnlycke, I. (2013). ***An approach to automating mobile application testing on Symbian Smartphones.*** KTH Information and Communication Technology,

[11] Crispin, L., Gregory, J., & Pearson Education, Inc. (2009). ***A Practical Guide For Testers And Agile Teams (1st. ed.).*** Boston, MA: Addison-Wesley. [47]

[12] Blé Jurado, C. (2010, enero). ***Diseño Ágil con TDD.*** Safe Creative, 301. [50]

[13] Vaca, P. A., Maldonado, C., Inchaurrondo, C., Peretti, J., Romero, M. S., & Bueno, M. (s.f.). Test-Driven Development. ***Una aproximación para entender su utilidad en el proceso de desarrollo de Software.,*** 10. [1]

[14] TutorialPoint. (2016). ***Agile Testing.*** Simply Easy Learning,45. [5]

[15] Penmetsa, J. R. (2017). ***Agile Testing.*** Springer Science+Business Media Singapore, 16. DOI :10.1007/978-981-10-1415-4_2. [1]

[16] S. Pressman, R. (2010). ***Ingeniería del Software un Enfoque Práctico*** (7ª ed.). México, D. F: McGraw-Hill. [12,185,239, 339,389,391-393]

[17] Myers, G. J., & Word Association, Inc. (2004). ***The Art of Software Testing*** (2ª ed.). Hoboken, New Jersey: John Wiley & Sons, Inc. [15]

[18] Scalone, F. (2006). ***Estudio Comparativo de los Modelos y Estándares de Calidad del Software.*** Facultad Regional de Buenos Aires: Universidad Tecnológica Nacional. [26]

[19] S. Wagner, (2013) ***Quality Models.*** Software Product Quality Control, Springer-Verlag Berlin Heidelberg. DOI 10.1007/978-3-642-38571-1 2. [34]

[20] Edvardsson, J. ***A Survey on Automatic Test Data Generation.*** Dept. of Computer and Information Science, Linkoping University, Sweden, 1-8

[21] Estayno, M., Dapozo, G., Cuenca, L., & Greiner, C. (s.f.). ***Modelos y Metricas para Evaluar Calidad de Software.*** Departamento de Informática. Facultad de Ciencias Exactas y Naturales y Agrimensura, Departamento de Ingeniería en Sistemas de Información. Facultad Regional Resistencia,6. [2]

[22] ***Automated Software Test Data Generation: Direction of Research*** (pp. 15-54).

[23] Boughton, A. ***Software Metrics.*** [3]

[24] McCall, J. A., Richards, P. K., and Walters, G. F., ***"Factors in Software Quality",*** Nat'l Tech.Information Service, [4,6]

[25] Giese, H. (s.f.). Software Engineering Group ***Software Quality Assurance: Introduction.*** University of Paderborn, 59. [32]

[26] Wadhwa, M. (2014). ***A Comparative Study of Software Quality Models.*** International Journal of Computer Science and Information Technologies, 5(4), 5. [2]

[27] Fillottrani, P. R. (2007). Calidad en el Desarrollo de Software ***Modelos de calidad de software. Universidad Nacional del Sur***, 19. [12]

[28] Alfonso, P. (2012). ***Revisión de modelos para evaluar la calidad de productos Web. Experimentación en portales bancarios del NEA.*** La Plata: Universidad Nacional de La Plata. [12]

[29] International Business Machines Corp. (2006). ***Eclipse Platform Technical Overview.*** 1-19.

[30] Hamill P., (2004). "O'Reilly Media, Inc.", ***Unit Test Frameworks: Tools for High-Quality Software Development.***

[31] J. A. Mera-Paz, ***"Análisis del proceso de pruebas de calidad de software"***, Ingeniería Solidaria, vol. 12, no. 20, pp. 163-176, oct. 2016. doi: http://dx.doi.org/10.16925/in.v12i20.1482 [7]

[32] González, L. (2009). ***Método Para Generar Casos de Prueba Funcional en el Desarrollo de Software.*** Revista Ingenierías Universidad de Medellín, 8(15), 29-36. [3]

[33] Aristegui, J. L. (2010). ***Los casos de prueba en la prueba de software.*** Revista Digital Lámpsakos, (3), 27-34. [3]

[34] Juristo, N., Moreno, A. M., & Vegas, S. (2006). ***Técnicas de Evaluación de Software.*** 131.[11,45,43]

[35] Pressman, R. S. (2001). ***Software Engineering a Practitioner's Approach*** (5ª ed.). New York, NY: McGraw-Hill Higher Education. [463-465,495]

[36] Hunt, A., & Thomas, D. (2003). ***Pragmatic Unit Testing in Java with JUnit***. ISBN 0-9745140-1-2

[37] Expósito, A. (2012). ***Pruebas de Caja Blanca***. Prueba y Mantenimiento del Software, 27. [3]

[38] Sommerville, I. (2005). ***Ingeniería del Software*** (7ª ed.). Madrid, España: Perarson Educación S.A. [85,501,511]

[39] Williams, L. (2006). ***White-Box Testing.*** [1-2]

[40] Massol, V., & Husted, T. (2004). ***JUnit in Action.*** Greenwich, CT: Manning Publications.

[41] Barber, S. (2006). ***Introduction to Performance Testing.*** PerfTestPlus, 25. [10]

[42] Manzor, S. ***Application Performance Testing Basics.*** Agileload, 13. [4,7]

[43] TutorialsPoint. ***Security Testing Protect the Daata and Maintain Functionality*** 134. [8]

[44] Esmite, I., Farias, M., Farias, N., & Perez, B. ***Automatización y Gestión de las Pruebas Funcionales usando Herramientas Open Source.*** Centro de Ensayos de Software (CES), 12. [2]

[45] Vishawjyoti, & Sharma, S. (2012). ***Study and Analysis of Automation Testing Techniques.*** Journal of Global Research in Computer Science, 3(12), 1-8. Recuperado de www.jgrcs.info. [1]

[46] Guideline for stresstest. ***Stress test*** (pp. 1-6) [2]

[47] Osfi-Bsif. (2009, diciembre). ***Stress Testing.*** Sound Business and Financial Practices, 1-12. [2]

[48] Guideline for volumen test. ***Volume test*** (pp. 1-2) [1]

[49] Rodríguez, E. (2011). ***Importancia de las pruebas de software.*** CINVESTAV, 1-42. [18]

[50] Rogério, P. (2007). ***Integration Testing.*** 1-25. [9,11,13,15]

[51] Paiva, A. (2007). ***GUI Testing.*** MFES, 1-22. [3.7]

[52] Horowitz, E., & Singhera, Z. A ***Graphical User Interface Testing Methodology.*** Department of Computer Science University of Southern California, 1-25. USC-CS-93-550. [1-3,17]

[53] Qureshi, I. A., & Nadeem, A. (2013). ***GUI Testing Techniques: A Survey.*** International Journal of Future Computer and Communication, 2(2), 1-5. [1]

[54] Tutorialspoint. ***Software Testing Software System Evaluation.*** TutorialsPoint, 32. [22]

[55] Farooq, S. U., & Quadri, S. M. K. (2011). ***3W's of Static Software Testing Techniques.*** Global Journal of Computer Science & Technology, 11(5), 1-11. ISSN: 0975-5861 [1]

[56] Vidal, J., Palacios, F., & Zambrano, F. (2013). ***Selenium Manual de Instalación y Uso.*** Universidad del Valle.

[57] Ishrat, M., Manish, S., & Alamgir, M. ***Comparison of Static and Dynamic Analysis for Runtime Monitoring.*** Mohd. Ishrat et al, International Journal of Computer Science & Communication, 2(5), 615-617. [2]

[58] Bhattad, B., & Kothari, A. (2014). ***Study od Defects, Testcases and Testing Challenge in Website Projects Using Manual and Automated Techniques.*** Computer Science & Information Technology (CS & IT), 11. DOI: 10.5121/csit.2014.4907 [9]

[59] Aguilera, M., & Gálvez, S. (2012). ***Análisis Sintáctico.*** Traductores, Compiladores e Intérpretes, 35. [2]

[60] Selenium. (2012). ***Selenium Documentation Release*** 1.0. Selenium.

[61] Bechtold, S., Brannen, S., Link, J., Merdes, M., Philipp, M., & Stein, C. JUnit 5 User Guide. Recuperado de https://junit.org/junit5/docs/current/user-guide/#dependency-diagram

[62] SeleniumHQ. (s.f.). SeleniumHQ Browser Automation. Recuperado de http://www.seleniumhq.org/

[63] Bergmann, S. (2015). ***PHPUnit Manual.*** Copyright ©, Sebastian Bergmann: Publication date Edition for PHPUnit 6.5.

[64] Villa, A., & Giraldo, J. (2012). ***Automatización de pruebas unitarias de códigos PHP.*** UAEM redaly, XVII (50), 147-150. Recuperado de http://www.redalyc.org/html/849/84923878021/

[65] Copyright © 2001-2018 PHP Group. (2018). Phar. Recuperado de http://php.net/manual/es/intro.phar.php

[66] Bergmann, S. (2018). PHPUnit. Recuperado de https://phpunit.readthedocs.io/en/latest/organizing-tests.html