# UNIVERSIDAD NACIONAL DE PIURA

## FACULTAD DE INGENIERÍA INDUSTRIAL

### ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA

**TESIS**

# "IMPLEMENTACIÓN DE UN GENERADOR DE CASOS DE PRUEBA PARA PROCEDIMIENTOS EN LENGUAJE JAVA"


**PRESENTADA POR:**

NAZARET TAYLER ALDAIR VIERA IPANAQUE


TESIS PARA OPTAR POR EL TITULO DE

INGENIERO INFORMATICO

LINEA DE INVESTIGACIÓN:

INFORMÁTICA, ELECTRÓNICA Y TELECOMUNICACIONES

SUB-LINEA DE INVESTIGACIÓN:

COMPUTACIÓN

PIURA, PERU

2019

TESIS PRESENTADA COMO REQUISITO PARA OPTAR POR EL TITULO DE

# INGENIERO INFORMATICO

ASESOR:

Ing. Víctor Hugo Valle Ríos

Magister en Administración de empresas.

Firma

TESISTA:

Nazaret Tayler Aldair Viera Ipanaque

Bachiller en Ingeniería informática

Firma

# DECLARACIÓN JURADA DE ORIGINALIDAD DE LA TESIS

Yo, **Nazaret Tayler Aldair Viera Ipanaque**, identificado con CU/DNI N° **75424335**, Bachiller de la Escuela Profesional de Ingeniería Informática de la Facultad de Ingeniería Industrial y domiciliado en Calle Libertad 313 del Distrito de Castilla, Provincia de Piura, Departamento de Piura, Celular 958610603 Email: nazaviera@gmail.com.

**DECLARO BAJO JURAMENTO:** que la tesis que presento es original e inédita, no siendo copia parcial ni total de una tesis desarrollada, y/o realizada en el Perú o en el Extranjero, en caso contrario de resultar falsa la información que proporciono, me sujeto a los alcances de lo establecido en el Art. N° 411, del código Penal concordante con el Art. 32° de la Ley N° 27444, y Ley del Procedimiento Administrativo General y las Normas Legales de Protección a los Derechos de Autor.

En fe de lo cual firmo la presente:

Piura, 12 de agosto de 2019.

NAZARET TAYLER ALDAIR VIERA IPANAQUE

DNI N° 75424335

Artículo 411.- El que, en un procedimiento administrativo, hace una falsa declaración en relación con hechos o circunstancias que le corresponde probar, violando la presunción de veracidad establecida por ley, será reprimido con pena privativa de libertad no menor de uno ni mayor de cuatro años.

Art. 4, Inciso 4.12 del reglamento del Registro Nacional de Trabajos de Investigación para optar por grados académicos y títulos profesionales – RENATI Resolución de Consejo Directivo N° 033-2016-SUNEDU/CD

# UNIVERSIDAD NACIONAL DE PIURA

## FACULTAD DE INGENIERÍA INDUSTRIAL

## ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA

# IMPLEMENTACIÓN DE UN GENERADOR DE CASOS DE PRUEBA PARA PROCEDIMIENTOS EN LENGUAJE JAVA.

**TESIS PARA OPTAR POR EL TÍTULO DE**

**INGENIERO INFORMÁTICO**

**APROBADA POR LOS JURADOS:**

DR. REUCHER CORREA MOROCHO
PRESIDENTE

DR. HUGO VICTOR ROSALES GARCÍA
SECRETARIO

ING. ARTURO SANDOVAL RIVERA
VOCAL

**UNIVERSIDAD NACIONAL DE PIURA**
**FACULTAD DE INGENIERÍA INDUSTRIAL**
**DECANATO**

# ACTA DE EVALUACIÓN Y SUSTENTACIÓN DE TESIS

## Expediente N° 1315 / 2016

Los miembros del Jurado Calificador Ad-Hoc de la Sustentación de Tesis nombrado con Resolución N° **648-CF-FII-UNP-16** de fecha 09/11/2016 que suscriben, se reunieron en acto público en la sala de exposiciones de la Facultad de Ingeniería Industrial de la Universidad Nacional de Piura, el día **07 de Agosto del 2019** a las **04:30 pm**, para evaluar la defensa de la Tesis titulada **"IMPLEMENTACIÓN DE UN GENERADOR DE CASOS DE PRUEBA PARA PROCEDIMIENTOS EN LENGUAJE JAVA."**, presentada por el Bachiller **NAZARET TAYLER ALDAIR VIERA IPANAQUE** y asesorado por el **MSc. VÍCTOR HUGO VALLE RÍOS** .

Después de haber calificado el Informe Final de la Tesis, escuchada la sustentación y las respuestas a las preguntas formuladas por el Jurado, se le declara APROBADO para optar el Título de **INGENIERO INFORMÁTICO** con el puntaje de 66.33 que corresponde al calificativo de BUENO.

| Jurado / Calificación | Presidente | Secretario | Vocal | Puntaje Promedio |
|---|---|---|---|---|
| Documento (Max 60 puntos) | 40 | 36 | 42 | 39.33 |
| Sustentación (Max 40 puntos) | 27 | 27 | 27 | 27.00 |
| PUNTAJE TOTAL | | | | 66.33. |

En consecuencia, el sustentante queda en condición de recibir el Título Profesional que se indica, conferido por el Consejo Universitario de la Universidad Nacional de Piura de conformidad con las Normas Estatutarias y la Ley Universitaria en vigencia.

FACULTAD DE INGENIERÍA INDUSTRIAL Dirección de Investigación * UNP *

Ciudad Universitaria, 07 de Agosto del 2019

| | | |
|---|---|---|
| Dr. REUCHER CORREA MOROCHO | Dr. HUGO VÍCTOR ROSALES GARCÍA | Ing. ARTURO SANDOVAL RIVERA |
| **PRESIDENTE** | **SECRETARIO** | **VOCAL** |

## Dedicatoria

El presente trabajo de investigación está dedicado a mi familia, los impulsadores de mi espíritu de superación. A mis amigos con quienes he tenido la dicha de compartir mi formación profesional y finalmente a los maestros que han marcado mi camino universitario y profesional.

**Agradecimiento**

Mi agradecimiento a la Universidad Nacional de Piura, a toda la Facultad de Ingeniería Industrial, a mis profesores quienes con la enseñanza de sus valiosos conocimientos hicieron que pueda crecer día a día como profesional, gracias a cada una de ustedes por su paciencia, dedicación, apoyo incondicional y amistad.

De igual manera, quiero expresar mis más grande y sincero agradecimiento al Ing. Víctor Valle Ríos, principal colaborador durante todo este proceso, quien con su dirección, conocimiento, enseñanza y colaboración permitió el desarrollo de este trabajo.

# Índice

## Índice de tablas

## Índice de figuras

**Resumen**

Las pruebas de software en su unidad más básica, las pruebas unitarias, suelen ser dejadas de lado al momento de programar un sistema de información, ya que estas emplean tiempo que puede ser usado para otras actividades y en cambio, la fase de pruebas se pospone hasta tener componentes desarrollados y funcionando, en donde realizar pruebas unitarias ya no ofrece ningún beneficio optando por realizar otras como pruebas funcionales o de integración, debido a esto, tener una herramienta que automatice dichas pruebas y que ayude a detectar errores en la fase de programación llega a ser de gran ayuda, debido a esto, se construyó una herramienta que realice dicha tarea, la cual utiliza algoritmos de búsqueda dispersa para calcular los posibles casos de prueba, y se estudió el posible impacto que tendría en la calidad del software desarrollado tomando como medida el número de errores encontrados. Los datos analizados sugieren que el uso de una herramienta como la desarrollada aumenta la calidad del software al encontrar más errores de los que se encuentran con desarrollos tradicionales.

Palabras clave: Generación de casos de prueba, Pruebas de software, Automatización.

**Abstract**

*The software tests in its most basic unit, the unit tests, are usually left aside when programming an information system, since they use time that can be used for other activities instead. The testing phase is postponed until having developed and functional components, where performing unit tests no longer offers any benefits opting for others like functional or integration tests. So having a tool that automates these tests and helps detect errors in the programming phase it becomes a great help. The created tool, uses dispersed search algorithms to calculate the possible test cases, and was studied the possible impact that it would have on the quality of the software developed, taking as a measure the number of errors found. The data analyzed suggests that the use of a tool such as the one developed increase the quality of the software by finding more errors than those found with traditional developments.*

*Keywords: Test cases generation, Software test, Automation.*

## Introducción

En los últimos años, el entorno en el que las compañías y empresas desarrollan sus actividades es cada vez más complejo, todos pugnan por tener una rentabilidad más alta, haciendo que, conocer a sus clientes, conocer gustos, preferencias y hábitos de compra sea una de las actividades más importantes para lograrlo, en este contexto los sistemas de información constituyen una gran herramienta de ayuda pues, simplifican el cálculo y procesan gran cantidad de datos en poco tiempo, permitiendo tomar decisiones con un nivel de certeza mayor.

A la hora de delimitar el concepto de sistema de información existe gran variedad de definiciones, la más precisa puede ser la propuesta por Andreu (1996), en la cual un sistema de información se entiende como: "conjunto formal de procesos que, operando sobre una colección de datos estructurada de acuerdo a las necesidades de la empresa, recopila, elabora y distribuyen selectivamente la información necesaria para la operación de dicha empresa y para las actividades de dirección y control correspondientes, apoyando, al menos en parte, los procesos de toma de decisiones necesarios para desempeñar funciones de negocio de la empresa de acuerdo con su estrategia".

Ahora bien, ¿Qué aspectos se deben tener en cuenta para lograr desarrollar uno de estos sistemas de información tan útiles? Según (Toledo, 2011): "Viéndolo en forma simplificada (o abstracta), los grandes rasgos a desarrollar de un sistema de información son:

Las estructuras de datos.

Las reglas de negocio (validaciones de datos, restricciones, etc.)

Los procedimientos que procesan estos datos y las pantallas que los muestran".

Entonces, ¿Cómo se prueba ello?, “para probar una funcionalidad se tienen que generar datos de prueba en forma manual, con código o por la interfaz de la aplicación, e incluso hay que pensar en las distintas situaciones que se intentan contemplar y ver que al ejecutar el proceso, se cumple lo que uno quería hacer” (Toledo, 2011). Realizar estas tareas suele ser tedioso por lo que, una herramienta que las automatice sería de gran utilidad en los desarrollos de software, pues aumentaría la cantidad de pruebas que se realizan al software, haciendo que encontrar errores sea más probable, con lo que se estará creando un software más robusto y confiable.

Por estos motivos las técnicas para la generación automática de casos de prueba suelen ser de gran ayuda pues, tratan de encontrar de forma eficiente un conjunto pequeño de casos de prueba que permitan cumplir un determinado criterio de suficiencia.

En el siguiente trabajo se desarrollará un generador de casos de pruebas automático, con el cual se estudiará si una herramienta como la descrita favorece o no de manera significativa la calidad de un software mientras este es desarrollado.

# Capítulo 1:

# El problema de investigación.

## 1.1. Descripción del problema de investigación.

Realizar pruebas de software es una tarea crucial y a la vez muy desafiante dentro del proceso de desarrollo de software, pues permite encontrar errores y problemas del software contra la especificación del mismo y cumple un rol fundamental en el aseguramiento de la calidad del producto, entre los tipos de pruebas que se pueden realizar al software están las pruebas de unidad, carga, rendimiento, estrés, integración y funcionales. Cada una de ellas tiene distintos objetivos y son realizadas en diferentes etapas del desarrollo del software. En el primer tipo mencionado, se desarrollan pruebas a componentes individuales de un sistema de software. Los desarrolladores especifican y codifican pruebas para cubrir todos o al menos una parte significativa de los posibles estados/configuraciones del artefacto o unidad de software, para simular el entorno del componente y descubrir la presencia de errores o "bugs". Dado que escribir todas esas pruebas de forma manual es costoso, las pruebas de unidad son generalmente realizadas de manera ineficiente o simplemente dejadas de lado. El panorama es aún peor, más allá del esfuerzo, porque las pruebas de software no pueden ser usadas para probar la ausencia de errores en el software sino tan solo la presencia. Por eso es necesario atacar el problema desde diferentes enfoques, cada uno teniendo sus fortalezas y ventajas. (Barrientos, 2014)

Realizar estas tareas suele ser tedioso, pues el número de casos de prueba necesarios para probar un programa software es relativamente infinito, por lo que es imposible conseguir un programa totalmente probado, siendo además el proceso de prueba muy costoso.

### 1.1.1. Formulación del problema

¿Cómo la implementación de un generador de casos de prueba mejorará la calidad de los procedimientos java que se implementan en el proceso de desarrollo de software?

### 1.1.2. Objetivos

#### *1.1.2.1.Objetivo General.*

- Implementar un generador de casos de prueba para procedimientos en lenguaje java.

#### *1.1.2.2.Objetivos Específicos*

- Definir los requisitos.
- Definir los sprints y backlogs necesarios.
- Construir los sprints definidos.

### 1.1.3. Justificación

Durante el desarrollo de software, cualquiera sea el enfoque usado o la metodología empleada, siempre hay una parte del mismo que se basa únicamente en validar y verificar que aquello que se ha hecho sea lo que el usuario quiere, y que mientras estos procesos son realizados, no surjan errores, esta parte del desarrollo de software es conocido como test o pruebas de software.

Existen varios tipos de pruebas, dependiendo de que parte del sistema de información en desarrollo se esté poniendo a prueba, o de lo que se busque al realizarlas, la prueba más básica que se realiza es consecuente con la parte más pequeña de software que se puede desarrollar, el componente; esta prueba es conocida como prueba de unidad o prueba unitaria y en ella lo que se busca es que el componente realice su tarea, independientemente de quien lo use o en que otros componentes se apoye para ejecutar

su tarea; cada componente, en las pruebas de caja blanca, bajo el enfoque del camino básico, posee una complejidad ciclomática, que es un valor que indica que tan complejo es el componente de probar y mantener; este valor junto al grafo de flujo genera distintos caminos, los cuales deben ser probados en su mayoría para asegurar que el software no falle en todos los casos; a veces, la cantidad de caminos básicos que se desprenden de este análisis, es demasiado grande como para ser probado de manera manual, puesto que, al ser las pruebas de software parte del desarrollo del mismo, el tiempo que se gaste en él significa un costo mayor para el producto final, por lo que una herramienta que nos permita probar la mayoría o totalidad de caminos básicos derivados de un componente sería de mucha utilidad, disminuyendo el tiempo empleado en la ejecución de pruebas de unidad y con ello el costo de la realización de estas pruebas, además de tener una mayor certeza de que el software tiene menos fallas ( recordemos que las pruebas de software se realizan para demostrar la presencia de los errores, no para demostrar que estos no existen) o también la estimación de los límites del software.

# Capítulo 2:

# Marco Teórico

## 2.1. Marco Referencial.

### 2.1.1. Las Empresas desarrolladoras de software en el Perú.

La industria de software es uno de los sectores más dinámicos y de continuo crecimiento a nivel mundial, en nuestro país la industria del software tiene un expectante potencial de crecimiento, logrando que las empresas peruanas desarrolladoras de software puedan convertirse en un competidor fuerte tanto internamente como en el mercado exterior.

Según (Pumarejo, 2002), las empresas desarrolladoras de software en el Perú producen software a medida, con pocos desarrollos genéricos. Entre las empresas locales, las 20 más grandes compañías tienen una participación de mercado del 90% de la producción doméstica.

El sector del software peruano se ve influenciado por factores competitivos como el precio, el servicio post venta y la calidad de los desarrollos; asimismo, la modernización de las entidades públicas y las empresas privadas, el creciente uso de Internet por empresas y hogares, el desarrollo de las telecomunicaciones conectadas a una red de computadoras y la competencia creciente en la economía globalizada, son factores que continuarán empujando hacia arriba la demanda por software peruano.

El interrogante radica en cómo sentar las bases para hacer del software una industria competitiva en el país. La implementación de sistemas de gestión de la calidad constituye una de las respuestas a esta interrogante.

### 2.1.2. Modelos de mejora de procesos de software

En esta sección se introducen conceptos de los diferentes modelos de mejora de procesos, tales como: IDEAL y Agile SPI.

#### *2.1.2.1. IDEAL*

MODELO IDEAL define un marco de ciclo de vida para la mejora de procesos, el cual proporciona un conjunto de actividades coherentes para sustentar la adopción de las prácticas recomendadas por el CMM. La forma en que dichas prácticas serán aplicadas tendrá variaciones de una entidad a otra, dependiendo por ejemplo del tipo de industria de software (ejemplos: software de proceso de datos, en tiempo real, etc.), del tamaño de la organización, de las modalidades de operación (ejemplos: si se adquiere software externo, si solamente se hace mantenimiento de productos existentes, etc.)

El Modelo IDEAL tiene como principal propósito establecer los mecanismos necesarios para facilitar la realización de programas de mejora continua en organizaciones, aunque está especialmente orientado a las mejoras en organizaciones intensivas en software. Este modelo de mejora define las siguientes fases para la realización satisfactoria de un programa de mejora de procesos:

1. Iniciación, cuyo propósito consiste en establecer los objetivos que se deberán alcanzar con la utilización sistemática de la mejora; en este caso, en el proceso de verificación y validación, desarrollar el plan para la realización de las mejoras (incluyendo la dotación de los recursos necesarios) y la obtención del compromiso requerido en cuanto a los objetivos, actividades, calendario y recursos disponibles para el programa de mejora.

2. Diagnóstico, cuyo objetivo consiste en establecer las prácticas eficientes ya existentes en la organización e identificar las necesidades y oportunidades

concretas; en este caso, relativas a la mejora de los procesos de verificación y validación.

3. Establecimiento, que persigue la definición del proceso y de todas las guías que permitan adaptar el proceso general definido a cada uno de los tipos de trabajos realizados por la organización.

4. Actuación, cuyo propósito es el logro de la utilización generalizada en todos los trabajos de la organización del proceso mejorado.

5. Aprendizaje tiene como propósito cuantificar las mejoras logradas con la introducción del nuevo proceso y determinar los próximos objetivos de mejora a satisfacer.

Dichas fases se encuentran representadas en la siguiente figura.

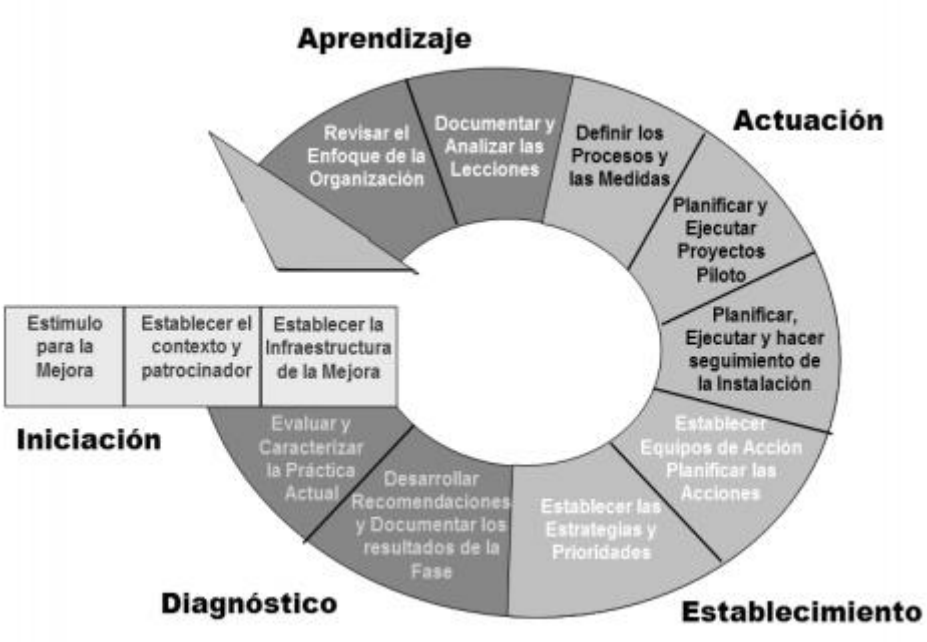



*Figura 1 Modelo IDEAL*

#### *2.1.2.2. Agile SPI.*

Según (J. Pino, 2003), Agile SPI es un marco de trabajo creado para la industria del software de Colombia, formada en gran parte por micro, pequeñas y medianas empresas - pymes.

Agile SPI es un framework de SPI (Software Process Improvement) que se caracteriza por:

- Guiar la mejora de los procesos de desarrollo de software, manteniendo el nivel de agilidad que la empresa desee.
- Ser un framework basado en modelos livianos para el soporte de un programa de mejoramiento continuo, a través de un proceso de mejora ágil.
- Estar acorde con una industria dinámica, creativa, innovadora e incierta como lo es la industria del software. Una industria donde el conocimiento y el talento humano son elementos fundamentales para garantizar su éxito.

Básicamente se ha formado su estructura a partir de los componentes primarios de un programa de mejora: una guía de mejora y unos modelos de soporte. En el caso de Agile SPI, los modelos son: modelo de calidad Agile SPI – Light Quality Model; el modelo de evaluación Agile SPI – Light Evaluation Model; y el modelo de métricas Agile SPI – Light Metrics Model.

Hay dos elementos integradores de toda la estructura: el modelo conceptual de soporte Framework PDS y el proceso que integra de manera dinámica los componentes Agile SPI – Process (Guía de mejoramiento). A continuación, presentamos la arquitectura de Agile SPI.

Descripción breve de los componentes del modelo integral de mejoramiento Agile SPI:

1. Un proceso ágil que guía a un proyecto de mejora de procesos en el marco de un programa de mejora, Agile SPI – Process. Es un proceso que cuenta con los elementos básicos para hacer posible que pymes puedan adelantar esfuerzos hacia la adecuación de un proceso de desarrollo acorde a sus necesidades. Este proceso es el marco de referencia para la gestión de los proyectos de mejora, y está integrado por el método, los modelos, la infraestructura, las técnicas y las herramientas de soporte.

2. Un modelo de calidad liviano, Agile SPI – Light Quality Model, que integra proceso y producto, y que guía la organización de las personas y los equipos, las disciplinas y las áreas de trabajo asociadas a la definición, aplicación y mejora del proceso hacia un nivel de madurez definido.

3. Un modelo de evaluación liviano, Agile SPI – Light Evaluation Model, que permite identificar y diagnosticar problemas de la industria en cuanto al proceso, y que permite trazar unos planes de mejora de acuerdo a un modelo/estándar de calidad definido.

4. Un método de evaluación ágil, Agile SPI - Process Assessment Method, el cual guía las actividades de evaluación distribuyendo todo su esfuerzo a lo largo de todo el proyecto de mejora. Esto lo aborda manejando dos intensidades de evaluación: superficial, la cual corresponde a la valoración con fines de diagnóstico inicial, y profundo y acotada, para fines de mejoramiento y verificación de mejoras antes de pasar a una entidad certificadora.

5. Un modelo de medida liviano, Agile SPI – Light Metrics Model, que permite medir el desempeño del proceso en los proyectos en los cuales es aplicado, mejorar las estimaciones de los proyectos a través de la medida del esfuerzo, la madurez de éste y la mejora del proceso en el marco de un programa SPI.

6. Un marco conceptual y tecnológico para la definición, visualización y aplicación de procesos, Agile SPI – Framework. Este marco conceptual se basa en el metamodelo SPEM – Software Process Engineering Metamodel, y este marco es la base conceptual sobre la cual se soportan todos los modelos de Agile SPI y las herramientas de soporte. Agile SPI – Framework permite relacionar los elementos del proceso con los elementos del modelo de calidad, con el modelo de evaluación y con el modelo de medida.

Una característica fundamental del Framework fue desarrollar con independencia los modelos presentes, de tal forma que fuera adaptable a las necesidades de la organización.

### 2.1.3. Área de testing

Las principales funciones del personal del área de pruebas son comúnmente las siguientes:

- Estimación y Planificación de las pruebas.
- Seguimiento y Reporte del progreso de las pruebas.
- Gestión de equipo de trabajo.
- Evaluar requerimientos.
- Gestión de Requerimiento de ambientes.
- Realizar análisis y registrar estimaciones.

- Definir plan de pruebas.
- Definir estrategias.
- Preparar informes de avance periódicos para Gerencia de Sistemas.
- Realizar el seguimiento del plan de pruebas.
- Ejecutar plan de pruebas.
- Coordinar las pruebas.
- Re planificar las tareas y pruebas cuando la situación lo requiera.
- Armado de condiciones de pruebas.
- Documentar y diseñar los casos de prueba.
- Control y seguimiento de la ejecución de las pruebas.
- Registrar evidencia de prueba.
- Realizar pruebas funcionales.
- Realizar pruebas de integración.
- Asegurar la conformidad de los requerimientos entregados.
- Controlar pendientes.
- Realizar seguimiento de incidentes.
- Reportar defectos.
- Conocimientos de base de datos SQL.
- Conocimiento de herramientas para la registración y seguimiento de casos de prueba.

- Conocimiento de herramientas para la registración y seguimiento de defectos.
- Conocimiento de herramientas de automatización.
- Conocimiento de herramientas para la gestión documental.

## 2.2. Bases Teóricas Científicas

### 2.2.1. Software

Según (EcuRed, s.f.) “software se refiere al equipamiento lógico o soporte lógico de una computadora digital, y comprende el conjunto de los componentes legales necesarios para hacer posible la realización de tareas específicas; en contraposición a los componentes físicos del sistema, llamados Hardware. Tales componentes lógicos incluyen, entre muchos otros, programas informáticos como Procesador de textos, que permite al usuario realizar todas las tareas concernientes a edición de textos; software de sistema, tal como un sistema operativo, que, básicamente, permite al resto de los programas funcionar adecuadamente, facilitando la interacción con los componentes físicos y el resto de las aplicaciones, también provee una interfaz para el usuario”.

Una definición más formal de software es la que aporta (IEEE Std, 1993) en su glosario virtual, en el que software que definido como: “Es el conjunto de los programas informáticos, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Considerando esta definición, el concepto de software va más allá de los programas de cómputo en sus distintos estados: código fuente, binario o ejecutable; también su documentación, datos a procesar e información de usuario forman parte del software: es decir, abarca todo lo intangible, todo lo "no físico" relacionado.

El término “software” fue usado por primera vez en este sentido por John W. Tukey en 1957. En las ciencias de la computación y la ingeniería de software, el software es toda la información procesada por los sistemas informáticos: programas y datos.

El concepto de leer diferentes secuencias de instrucciones desde la memoria de un dispositivo para controlar los cálculos fue introducido por Charles Babbage como parte de su máquina diferencial. La teoría que forma la base de la mayor parte del software moderno fue propuesta por vez primera por Alan Turing en su ensayo de 1936, "Los números computables", con una aplicación al problema de decisión.

### 2.2.2. Clasificación de software

#### *2.2.2.1.Software de sistema*

Su objetivo es desvincular adecuadamente al usuario y al programador de los detalles de la computadora en particular que se use, aislándolo especialmente del procesamiento referido a las características internas de: memoria, discos, puertos y dispositivos de comunicaciones, impresoras, pantallas, teclados, etc.

#### *2.2.2.2.Software de programación*

Es el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Incluye entre otros: Editores de texto, compiladores, intérpretes, enlazadores, depuradores, entornos de desarrollo integrados, etc.

#### *2.2.2.3.Software de aplicación*

Es aquel que permite a los usuarios llevar a cabo una o varias tareas específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido, con especial énfasis en los negocios.

#### *2.2.2.4.Software social*

El software social no son propiamente aspectos de programación. Estas herramientas engloban correo electrónico, listas de correo electrónico, IRC, mensajería instantánea, bitácoras de red entre otros.

### 2.2.3. Sistemas de información

Conjunto formal de procesos que, operando sobre una colección de datos estructurada de acuerdo a las necesidades de la empresa, recopila, elabora y distribuyen selectivamente la información necesaria para la operación de dicha empresa y para las actividades de dirección y control correspondientes, apoyando, al menos en parte, los procesos de toma de decisiones necesarios para desempeñar funciones de negocio de la empresa de acuerdo con su estrategia (Andreu, 1996).

### 2.2.4. Modelos para el desarrollo de software

Un modelo para el desarrollo de software es una representación abstracta de un proceso. Cada modelo representa un proceso desde una perspectiva particular y así proporcione información parcial sobre el proceso. Éstos modelos generales no son descripciones definitivas de los procesos del software más bien son abstracciones de los procesos que se pueden utilizar para el desarrollo del software. Puede pensarse en ellos como marcos de trabajo del proceso y que pueden ser adaptados para crear procesos más específicos.

### 2.2.5. Metodologías de desarrollo de software

Es un enfoque estructurado para el desarrollo de software que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos. A lo largo del tiempo, una gran cantidad de métodos han sido desarrollados diferenciándose por su fortaleza y debilidad. El framework para metodología de desarrollo de software consiste en:

- Una filosofía de desarrollo de programas de computación con el enfoque del proceso de desarrollo de software.
- Herramientas, modelos y métodos para asistir al proceso de desarrollo de software.

Estos frameworks son a menudo vinculados a algún tipo de organización, que además desarrolla, apoya el uso y promueve la metodología. La metodología es a menudo documentada en algún tipo de documentación formal. Entre las metodologías de software conocidas están:

- Rational Unified Process (RUP)
- EXtreme Programming (XP)
- SCRUM
- Dynamic Systems Development Method (DSDM)
- Crystal Clear
- Lean Development (LD)
- Adaptive Software Development (ASD)

### 2.2.6. Metodología de desarrollo de software: SCRUM

#### *2.2.6.1.¿Qué es SCRUM?*

SCRUM es una metodología ágil y flexible para gestionar el desarrollo de software, cuyo principal objetivo es maximizar el retorno de la inversión para su empresa (ROI). Se basa en construir primero la funcionalidad de mayor valor para el cliente y en los principios de inspección continua, adaptación, auto-gestión e innovación (SOFTENG).

#### *2.2.6.2.¿Cuándo utilizar SCRUM?*

Con la metodología SCRUM el cliente se entusiasma y se compromete con el proyecto dado que lo ve crecer iteración a iteración. Asimismo, le permite en cualquier momento realinear el software con los objetivos de negocio de su empresa, ya que puede introducir cambios funcionales o de prioridad en el inicio de cada nueva iteración sin ningún problema. Esta metódica de trabajo promueve la innovación, motivación y compromiso del equipo que forma parte del proyecto, por lo que los profesionales encuentran un ámbito propicio para desarrollar sus capacidades.

#### *2.2.6.3.Beneficios de utilizar SCRUM*

Los beneficios de usar SCRUM son muchos, entre los principales tenemos:

- Cumplimento de expectativas: El cliente establece sus expectativas indicando el valor que le aporta cada requisito / historia del proyecto, el equipo los estima y con esta información el Product Owner establece su prioridad. De manera regular, en las demos de Sprint el Product Owner comprueba que efectivamente los requisitos se han cumplido y transmite se feedback al equipo.

- Flexibilidad a cambios: Alta capacidad de reacción ante los cambios de requerimientos generados por necesidades del cliente o evoluciones del mercado. La metodología está diseñada para adaptarse a los cambios de requerimientos que conllevan los proyectos complejos.

- Reducción del Time to Market: El cliente puede empezar a utilizar las funcionalidades más importantes del proyecto antes de que esté finalizado por completo.

- Mayor calidad del software: La metódica de trabajo y la necesidad de obtener una versión funcional después de cada iteración, ayuda a la obtención de un software de calidad superior.

- Mayor productividad: Se consigue entre otras razones, gracias a la eliminación de la burocracia y a la motivación del equipo que proporciona el hecho de que sean autónomos para organizarse.

- Maximiza el retorno de la inversión (ROI): Producción de software únicamente con las prestaciones que aportan mayor valor de negocio gracias a la priorización por retorno de inversión.

- Predicciones de tiempos: Mediante esta metodología se conoce la velocidad media del equipo por sprint (los llamados puntos historia), con lo que consecuentemente, es posible estimar fácilmente para cuando se dispondrá de una determinada funcionalidad que todavía está en el Backlog.

- Reducción de riesgos: El hecho de llevar a cabo las funcionalidades de más valor en primer lugar y de conocer la velocidad con que el equipo avanza en el proyecto, permite despejar riesgos eficazmente de manera anticipada.

#### *2.2.6.4.El proceso de desarrollo en SRUM*

El desarrollo se realiza de forma iterativa e incremental. Cada iteración, denominada Sprint, tiene una duración preestablecida de entre 2 y 4 semanas, obteniendo como resultado una versión del software con nuevas prestaciones listas para ser usadas. En cada nuevo Sprint, se va ajustando la funcionalidad ya construida y se añaden nuevas prestaciones priorizándose siempre aquellas que aporten mayor valor de negocio.

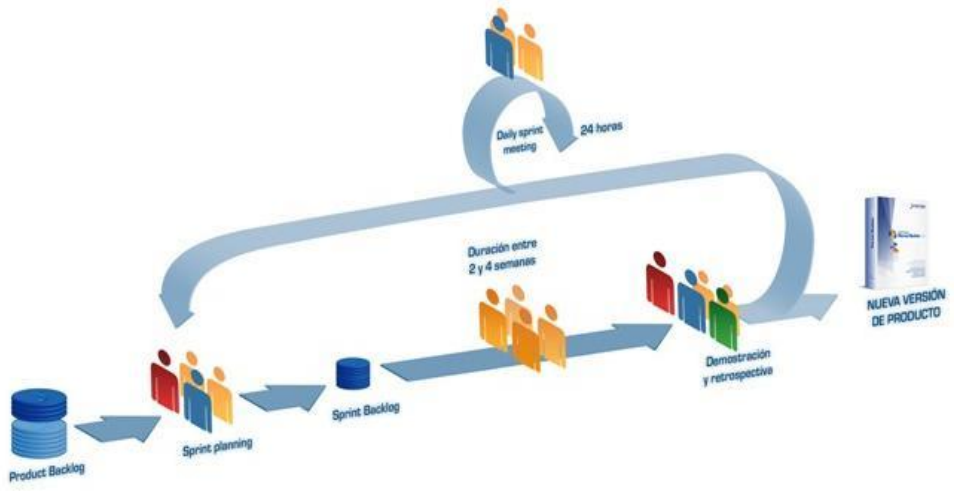



*Figura 2 Modelo de proceso de la metodología SCRUM*

#### 2.2.6.4.1. *Product Backlog:*

Conjunto de requisitos denominados historias, descritos en un lenguaje no técnico y priorizados por valor de negocio, o lo que es lo mismo, por retorno de inversión considerando su beneficio y coste. Los requisitos y prioridades se revisan y ajustan durante el curso del proyecto a intervalos regulares.

#### 2.2.6.4.2. *Sprint Planning:*

Reunión durante la cual el Product Owner presenta las historias del backlog por orden de prioridad. El equipo determina la cantidad de historias que puede comprometerse a completar en ese sprint, para en una segunda parte de la reunión, decidir y organizar cómo lo va a conseguir.

#### 2.2.6.4.3. *Sprint:*

Es un intervalo de tiempo de máximo 4 semanas, durante la cual el equipo trabaja para convertir las historias del Product Backlog a las que se ha comprometido, en una nueva versión del software totalmente operativo.

#### 2.2.6.4.4. *Sprint Backlog:*

Son la lista de las tareas necesarias para llevar a cabo el desarrollo del sprint.

- Daily sprint meeting: Reunión diaria de cómo máximo 15 min. en la que el equipo se sincroniza para trabajar de forma coordinada. Cada miembro comenta que hizo el día anterior, que hará hoy y si hay impedimentos.

- Demo y retrospectiva: Reunión que se celebra al final del sprint y en la que el equipo presenta las historias conseguidas mediante una demonstración del producto. Posteriormente, en la retrospectiva, el equipo analiza qué se hizo bien, qué procesos serían mejorables y discute acerca de cómo perfeccionarlos.

#### ***2.2.6.5.Roles***

En SCRUM, el equipo se focaliza en construir software de calidad. La gestión de un proyecto SCRUM se centra en definir cuáles son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en vencer cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo.

El equipo SCRUM está formado por los siguientes roles:

1. SCRUM master: Persona que lidera al equipo guiándolo para que cumpla las reglas y procesos de la metodología. Gestiona la reducción de impedimentos del proyecto y trabaja con el Product Owner para maximizar el ROI.

2. Product owner (PO): Representante de los accionistas y clientes que usan el software. Se focaliza en la parte de negocio y él es responsable del ROI del proyecto (entregar un valor superior al dinero invertido). Traslada la

visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el Product Backlog y las re prioriza de forma regular.

3. Team: Grupo de profesionales con los conocimientos técnicos necesarios y que desarrollan el proyecto de manera conjunta llevando a cabo las historias a las que se comprometen al inicio de cada sprint

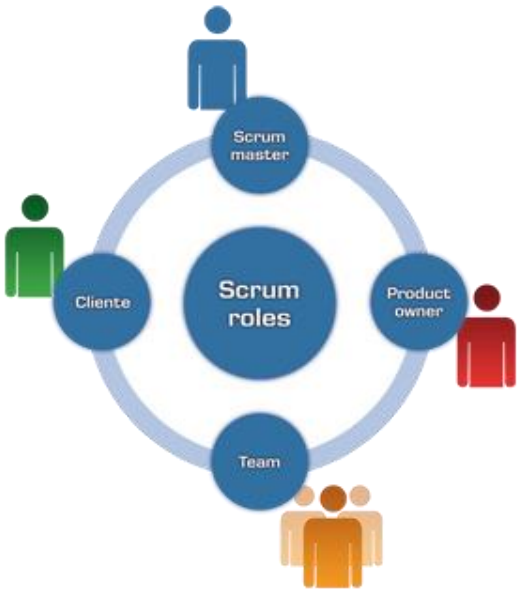



*Figura 3 Roles de SCRUM*

### 2.2.7. Pruebas de software

Las pruebas de software consisten en la dinámica de la verificación del comportamiento de un programa en un conjunto finito de casos de prueba, debidamente seleccionados de por lo general infinitas ejecuciones de dominio, contra la del comportamiento esperado. Son una serie de actividades que se realizan con el propósito de encontrar los posibles fallos de implementación, calidad o usabilidad de un programa u ordenador; probando el comportamiento del mismo (EcuRed, 2013).

#### *2.2.7.1. Objetivos de las pruebas de software*

La prueba de software es un elemento crítico para la garantía del correcto funcionamiento del software. Entre sus objetivos están:

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

Para lograr los objetivos propuestos, un ingeniero de software deberá conocer los principios básicos que guían las pruebas del software.

#### *2.2.7.2.Principios de las pruebas de software*

- Las pruebas se rigen por una serie de principios, una buena comprensión de estos facilitará el posterior uso de los métodos en un efectivo diseño de casos de prueba. A continuación, se citan: La prueba puede ser usada para mostrar la presencia de errores, pero nunca su ausencia.
- La principal dificultad del proceso de prueba es decidir cuándo parar.
- Evitar casos de pruebas no planificados, no reusables y triviales a menos que el programa sea verdaderamente sencillo.

- Una parte necesaria de un caso de prueba es la definición del resultado esperado.
- Los casos de pruebas tienen que ser escritos no solo para condiciones de entrada válidas y esperadas sino también para condiciones no válidas e inesperadas.
- El número de errores sin descubrir es directamente proporcional al número de errores descubiertos.

#### *2.2.7.3. Etapas involucradas en las pruebas de software*

- Seleccionar qué es lo que debe medir la prueba, es decir, cuál es su objetivo, para qué exactamente se hace la prueba.
- Decidir cómo se va a realizar la prueba, es decir, qué clase de prueba se va a utilizar para medir la calidad y qué clase de elementos de prueba se deben usar.
- Desarrollar los casos de prueba. Un caso de prueba es un conjunto de datos o situaciones de prueba que se utilizarán para ejecutar la unidad que se prueba o para revelar algo sobre el atributo de calidad que se está midiendo.
- Determinar cuáles deberían ser los resultados esperados de los casos de prueba y crear el documento que los contenga.
- Ejecutar los casos de prueba.

#### *2.2.7.4. Evaluación de resultados*

Comparar los resultados de la prueba con los resultados esperados. Cualquier discrepancia entre ellos significa un error. Típicamente el error está en el sistema o unidad

probada, pero también puede ser generado por algún aspecto del mismo proceso de prueba.

### 2.2.8. Pruebas de caja blanca

Las pruebas de caja blanca se basan en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de la caja blanca el ingeniero del software puede obtener casos de prueba que:

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican al software, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad (EcuRed, 2012).

### 2.2.9. Camino básico

La prueba del camino básico es una técnica de prueba de la Caja Blanca propuesta por Tom McCabe.

Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes

se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

- A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- Se calcula la complejidad ciclomática del grafo.
- Se determina un conjunto básico de caminos independientes.
- Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

### 2.2.10. Notación de Grafo de Flujo

Para aplicar la técnica del camino básico se debe introducir una sencilla notación para la representación del flujo de control, el cual puede representarse por un Grafo de Flujo. Cada nodo del grafo corresponde a una o más sentencias de código fuente. Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo. Para construir el grafo se debe tener en cuenta la notación para las instrucciones. Un Grafo de Flujo está formado por 3 componentes fundamentales que ayudan a su elaboración, comprensión y nos brinda información para confirmar que el trabajo se está haciendo adecuadamente. Los componentes son:

1. Nodo: Cada círculo representado se denomina nodo del Grafo de Flujo, el cual representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de

decisión. Puede ser también que hallan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.

2. Aristas: Las flechas del grafo se denominan aristas y representan el flujo de control, son análogas a las representadas en un diagrama de flujo. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

3. Regiones: Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran. La cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa.

### 2.2.11. Complejidad Ciclomática

La complejidad ciclomática es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente.

### 2.2.12. Derivación de casos de prueba

Luego de tener elaborados los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se

escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

Casos de prueba para cada camino. Camino 1: 1-2-3-5-6. Escoger algún X y Y tales que cumpla X >= 0 AND Y >= 0. X = 10 AND Y = 20. Camino 2: 1-2-4-6. Escoger algún X tal que se cumpla X < 0. X = -15.

Luego de confeccionar los casos de prueba se ejecutan cada uno de estos y se comparan los resultados con los esperados. Una vez terminados todos los casos de prueba, se estará seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez. Es importante considerar que algunos caminos no se pueden probar de forma aislada. O sea, la combinación de datos requeridos para recorrer el camino no se puede obtener con el flujo normal del programa. En tales casos, estos caminos se prueban como parte de otra prueba de camino.

### 2.2.13. Calidad de Software

El concepto de calidad es bastante genérico y abstracto, muchos autores nos brindan su punto de vista; (Pressman, 2010) nos dice que “en el sentido más general se define como un proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan”.

La definición anterior sirve para enfatizar tres puntos:

1. Un proceso eficaz de software establece la infraestructura que da apoyo a cualquier esfuerzo de elaboración de un producto de software de alta calidad. Los aspectos de administración del proceso generan las verificaciones y equilibrios que ayudan a evitar que el proyecto caiga en el caos, contribuyente clave de la mala calidad. Las prácticas de ingeniería

de software permiten al desarrollador analizar el problema y diseñar una solución sólida, ambas actividades críticas de la construcción de software de alta calidad. Por último, las actividades sombrilla, tales como administración del cambio y revisiones técnicas, tienen tanto que ver con la calidad como cualquier otra parte de la práctica de la ingeniería de software.

2. Un producto útil entrega contenido, funciones y características que el usuario final desea; sin embargo, de igual importancia es que entrega estos activos en forma confiable y libre de errores. Un producto útil siempre satisface los requerimientos establecidos en forma explícita por los participantes. Además, satisface el conjunto de requerimientos (por ejemplo, la facilidad de uso) con los que se espera que cuente el software de alta calidad. Al agregar valor para el productor y para el usuario de un producto, el software de alta calidad proporciona beneficios a la organización que lo produce y a la comunidad de usuarios finales. La organización que elabora el software obtiene valor agregado porque el software de alta calidad requiere un menor esfuerzo de mantenimiento, menos errores que corregir y poca asistencia al cliente. Esto permite que los ingenieros de software dediquen más tiempo a crear nuevas aplicaciones y menos a repetir trabajos mal hechos. La comunidad de usuarios obtiene valor agregado porque la aplicación provee una capacidad útil en forma tal que agiliza algún proceso de negocios. El resultado final es 1) mayores utilidades por el producto de software, 2) más rentabilidad cuando una aplicación apoya un proceso de negocios y 3) mejor disponibilidad de información, que es crucial para el negocio.

3. Un producto de alta calidad requiere menos mantenimiento y facilita tanto el desarrollo como el mantenimiento de la productividad. Con la medición de la calidad se pueden lograr estos objetivos. En lo que se refiere al mantenimiento, la medición de la calidad del software ayuda a identificar problemas de confiabilidad y a mejorar las técnicas para identificar las necesidades de mantenimiento (Pressman, 2010).

### 2.2.14. La técnica de Búsqueda Dispersa

La Búsqueda Dispersa (Scatter Search en inglés) es un método evolutivo que opera sobre un conjunto de soluciones, llamado Conjunto de Referencia (RefSet). Las soluciones presentes en este conjunto son combinadas con el fin de generar nuevas soluciones que mejoren a las originales. Así, el conjunto de referencia almacena las mejores soluciones que se encuentran durante el proceso de búsqueda, considerando para ello su calidad y la diversidad que aportan al mismo. Esta técnica utiliza estrategias sistemáticas para avanzar en el proceso de búsqueda en vez de aleatorias, siendo ésta una de las principales diferencias con los ampliamente utilizados Algoritmos Genéticos (Blanco, Díaz, & Tuya, 2006).

El algoritmo Scatter Search comienza generando un conjunto P de soluciones diversas mediante un método de generación de diversidad. Las soluciones presentes en este conjunto pueden ser mejoradas con un método de mejora, el cual es opcional.

Posteriormente se construye el conjunto de referencia con las mejores soluciones de P y las más diversas a las ya incluidas. A continuación, comienza un proceso cíclico en el cual el algoritmo crea subconjuntos del conjunto de referencia, con un método de generación de subconjuntos, y aplica un método de combinación sobre dichos subconjuntos para obtener las nuevas soluciones. Posteriormente, sobre cada nueva solución aplica un método de mejora y evalúa si debe incorporarse al conjunto de

referencia, mediante un método de actualización. El algoritmo se detiene cuando no se generan nuevas soluciones en el proceso de combinación

## 2.3. Antecedentes

1. **Automatización y Gestión de las Pruebas Funcionales usando Herramientas Open Source por Ignacio Esmite, Mauricio Farías, Nicolás Farías, Beatriz Pérez (2007)**

En este artículo se presenta una metodología y el conjunto de herramientas open source utilizadas para la automatización de las pruebas funcionales de productos con interfaz web. Se describe la experiencia de utilizar la metodología en un proyecto de automatización específico y se concluye la factibilidad para la automatización de las pruebas siguiendo las actividades y el conjunto de herramientas definidos.

En el proceso de automatización se utilizan tecnologías open source como lo son: Selenium, Eclipse y extensiones de Mozilla Firefox (Firebug, XPath Checker y XPather).

De este artículo se tomarán las herramientas y lógica usadas para realizar pruebas automatizadas logrando así mejorar el proceso de realización y ejecución de pruebas en el generador.

2. **Generación automática de casos de prueba mediante búsqueda dispersa por Raquel Blanco, Eugenia Díaz, Javier Tuya (2006)**

En este artículo se presenta un método de generación de pruebas basado en Búsqueda Dispersa que permite generar automáticamente casos de prueba según un criterio de suficiencia para obtener cobertura de ramas.

Para el desarrollo del método se utiliza búsqueda dispersa y los conceptos de distancias asociadas a una solución, pruebas de caja blanca, rango de prueba y cobertura de las pruebas.

Los resultados asociados muestran que este método se comporta mejor con rangos pequeños, debido a la utilización de la función de diversidad, existen otros métodos basados en búsquedas locales que cubren los nodos más difíciles, pero con una mayor cantidad de procesamiento.

De este artículo se tomará la técnica de generación de casos de prueba utilizada, la búsqueda dispersa, con la que se creará el algoritmo que genere dichos casos.

### 3. Generación automática de casos de prueba para Líneas de Producto de Software por Beatriz Pérez Lamancha (2009)

En este artículo, los casos de prueba generados, se logran a partir de Líneas de Producto de Software en las cuales se requiere definir los mecanismos para gestionar la variabilidad en las pruebas y su trazabilidad a los demás artefactos de desarrollo. Asimismo, los casos de prueba se generan automáticamente mediante el lenguaje de transformación QVT a partir de diagramas de secuencia extendidos para representar la variabilidad en la LPS. La trazabilidad entre los distintos modelos es gestionada mediante la definición de un Perfil de UML para el Modelo de Variabilidad Ortogonal.

De este trabajo se extrae las ideas de líneas de producto de software (LPS) que sirve para definir ciertas tareas, servicios y en general algoritmos bajo ciertas características comunes, las cuales satisfacen las necesidades específicas de un dominio o segmento particular del mercado.

4. **Generación automática de objetivos de prueba a partir de casos de uso mediante partición de categorías y variables operacionales por Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Jesús Torres, Arturo Torres-Zenteno (2007)**

Este trabajo complementa y amplia trabajos anteriores sobre generación de pruebas a partir de casos de uso presentando un proceso que, de manera sistemática y automática, permite generar objetivos de prueba a partir de casos de uso especificados en un lenguaje no formal. Este proceso aplica el método de categoría-partición y el patrón Use Case Test Pattern, el cual usa variables operacionales. Además, se presenta los algoritmos necesarios para la automatización del proceso propuesto.

De este trabajo tomaré prestado el proceso de generación de casos de prueba usando variables operacionales y los algoritmos necesarios para su realización.

5. M**ejora del proceso software de una pequeña empresa desarrolladora de software: caso competisoft-perú lambda por Dianne Britt Vergara González (2008)**

Este trabajo amplía el enfoque de mejora de procesos para el que se realiza el presente trabajo de investigación, permitiendo evaluar en el futuro que tanto se mejoró con un generador de casos de pruebas automático con respecto a pruebas manuales.

## 2.4. Hipótesis de la investigación

### 2.4.1. Formulación de la hipótesis.

- Con la implementación de un generador de casos de prueba para procedimientos en lenguaje java se conseguirá encontrar una mayor cantidad de errores que al subsanarlos, aumentarán la calidad del software desarrollado.

### 2.4.2. Identificación de variables.

#### *2.4.2.1. Variables independientes:*

- Generador de casos de prueba para procedimientos en lenguaje java.

#### *2.4.2.2. Variables dependientes:*

- Errores encontrados.

# Capítulo 3:

# Marco Metodológico

## 3.1. Diseño de la investigación

### 3.1.1. Diseño experimental

Para la comprobación de la hipótesis planteada se realizó una medición de las fallas encontradas cuando se utiliza la herramienta de software que se desarrolló y otra medición sobre la misma variable con datos obtenidos a través de una encuesta que se realizó a algunas empresas locales y otras de otras ciudades escogidas aleatoriamente y que no tuvieron relación alguna con el desarrollo de la investigación.

### 3.1.2. Metodología para el desarrollo de la herramienta informática.

Para la realización de la herramienta informática propuesta se usó una metodología de desarrollo ágil llamada SCRUM, está metodología se basa en un ciclo de vida iterativo e incremental donde todo el desarrollo del producto se divide en SPRINTS que son bloques de tiempo de un mes o menos durante el cual se da un incremento de producto terminado, utilizable y potencialmente desplegable.

### 3.1.3. Definición de requisitos

Para planificar y desarrollar los Sprints, primero se debe conocer lo que se necesita construir, por eso, el primero paso fue definir los requisitos de la herramienta:

**R1:** La herramienta debe permitir la ejecución de pruebas, brindando el código fuente el procedimiento a probar y generando los “n” casos de pruebas requeridos para obtener la mayor cantidad de fallas posibles.

**R2:** La herramienta debe permitir la ejecución de pruebas, brindando una carpeta de archivos de código fuente Java de la cual se tomarán todos los archivos de

código fuente Java que se encuentren y de cada uno de ellos se crearán casos de prueba para cada método que se encuentre.

**R3:** Muestre el código de pruebas que se genera para probar el procedimiento

**R4:** Agrupe resultados por casos de pruebas generados.

**R5:** Personalizar la configuración de la herramienta, cambiando configuraciones generales del generador que afecten de manera directa a la ejecución y resultados que se obtengan.

### 3.1.4. Aplicación de la metodología SCRUM

**Sprint y Backlogs**

La construcción de la herramienta necesaria para la investigación se dividió en 2 Sprints, el primero relacionado a la construcción del núcleo del programa que ejecutará las pruebas como tal, y el segundo referido más a la interfaz gráfica y experiencia de usuario incluyendo las pruebas que determinen que el software trabaja apropiadamente.

Para estimar el desarrollo de las historias de usuario se usó puntos historia que, muy resumidamente, son una unidad usada, normalmente, para medir el "tamaño" de una historia de usuario. Más concretamente, un punto historia es una fusión de la cantidad de esfuerzo que supone desarrollar la historia de usuario, la complejidad de su desarrollo y el riesgo inherente (Navarro & Garzás, 2015).

La experiencia, y varios estudios, han demostrado que es mejor estimar usando rangos o escalas fijas y conocidos de posibles valores que se pueden asignar a la estimación, es decir, fijando un tope máximo de puntos historia.

Problemas de tipo a usar un rango grandísimo de posibles puntos historia, como, por ejemplo, los enteros del 1 al 100 y entrar en debates interminables sobre si una historia debe tener 38 o 39 puntos historia.

Por esta razón se utilizó la secuencia de Fibonacci, la cual es útil porque la separación entre los números de la secuencia se hace más grande a medida que los números aumentan y fijando un límite máximo de 13.

Pero, ¿Por qué utilizar Puntos Historia para estimar y no horas? Una analogía para estos casos es comparar los Puntos Historia con la duración temporal (de una tarea) utilizando el símil del viaje con niños en el coche, en el que nos suelen preguntar: ¿Cuánto falta para llegar? Y, realmente, con exactitud, no sabes que responder. El caso es que tú sabes la distancia, pero no puedes predecir si habrá retenciones o paradas inesperadas, así que te limitas a contestar "Si no pasa nada, estaremos allí en, más o menos, X horas". La distancia entre dos ciudades puede ser la misma, pero el tiempo que se tarda en ir de una a otra puede variar.

Trasladándonos a un proyecto ágil, la distancia serían los Puntos Historia, los kilómetros que más o menos sabemos que hay de aquí a la ciudad a la que vamos. La distancia siempre va a ser la misma, es decir, es constante, pero las horas varían del momento de día en el que salga, y sus atascos, las fechas, etc.

*Tabla 1* Sprint 1: Desarrollo del núcleo del generador de casos de pruebas

| **N°** | **Historia** | **Puntos historia** |
|---|---|---|
| 1 | Como usuario necesito que se ejecuten pruebas de manera automática cuando brindo el código fuente del procedimiento a probar | 8 |
| 2 | Como usuario necesito que se ejecuten pruebas de manera automática cuando brindo una carpeta de archivo llena de archivos de código fuente java. | 13 |
| 3 | Como usuario necesito poder cambiar el código fuente que brinde para un procedimiento tomándolo como dato histórico | 5 |
| 4 | Como usuario necesito ver el código fuente de las pruebas que se generan al iniciar el proceso de ejecución de la herramienta | 5 |

*Tabla 2* Sprint 2: Desarrollo de la interfaz gráfica del generador

| N° | Historia | Puntos historia |
|---|---|---|
| 1 | Como usuario necesito poder entrar a una opción de la herramienta de manera intuitiva usando tanto el mouse como los accesos rápidos vía teclado | 1 |
| 2 | Como usuario necesito poder cambiar entre ventanas de manera sencilla sin necesidad de estar cerrando una para entrar a la siguiente | 1 |
| 3 | Como usuario necesito que los gráficos estadísticos sean lo más sencillos de comprender posible, quitando datos innecesarios y solo mostrando la información requerida | 3 |
| 4 | Como usuario necesito poder configurar el entorno de la herramienta, sean configuraciones básicos como de la generación de los casos de pruebas | 5 |
| 5 | Como usuario necesito que los resultados mostrados sean acertados. | 5 |

### 3.1.5. Construcción de los Backlogs definidos

Para la construcción de los Backlogs definidos, se necesita explicar de manera breve cómo funciona el núcleo de la herramienta desarrollada y como está compuesta internamente.

Primero, se explica el procedimiento operativo, el cual inicia cuando el usuario de la herramienta carga el código fuente que desea probar, escogiendo el procedimiento que se pondrá a prueba, seguido de ello y muy importante es iniciar la ejecución como tal, con lo que la herramienta calculará la complejidad ciclomática y con ella, se creará un conjunto con los caminos para los casos de prueba, estos casos de prueba serán sometidos al algoritmo de búsqueda dispersa en el cual se obtendrá un subconjunto mejorado de soluciones, para cada solución se crearán dominios para los datos de entrada con lo cual habremos terminado la parte de generación de casos de prueba, luego de ello se requieren los resultados de los mismos, a los cuales se llegará en dos pasos, el primero es crear código ejecutable mediante pruebas JUNIT y el segundo el someter dicho código a ejecución obteniendo los resultados que se presenten.

Para ello se ha desarrollado el algoritmo que se muestra a continuación:

1. *Cargar carpeta con el código fuente*
2. *Si archivo*
    a. *Escoger procedimiento a probar*
    b. *Compilar archivo*
3. *Si carpeta*
    a. *Escoger procedimiento y archivo a probar*
    b. *Compilar proyecto*
4. *Calcular complejidad ciclomática*
5. *Calcular caminos*
6. *Aplicar algoritmo de búsqueda dispersa para obtener los casos de prueba*
7. *Crear JUNIT Test para cada caso de prueba*
8. *Ejecutar pruebas*
9. *Extraer resultados*
10. *Mostrar resultados*

A continuación, el esquema algorítmico en una imagen:

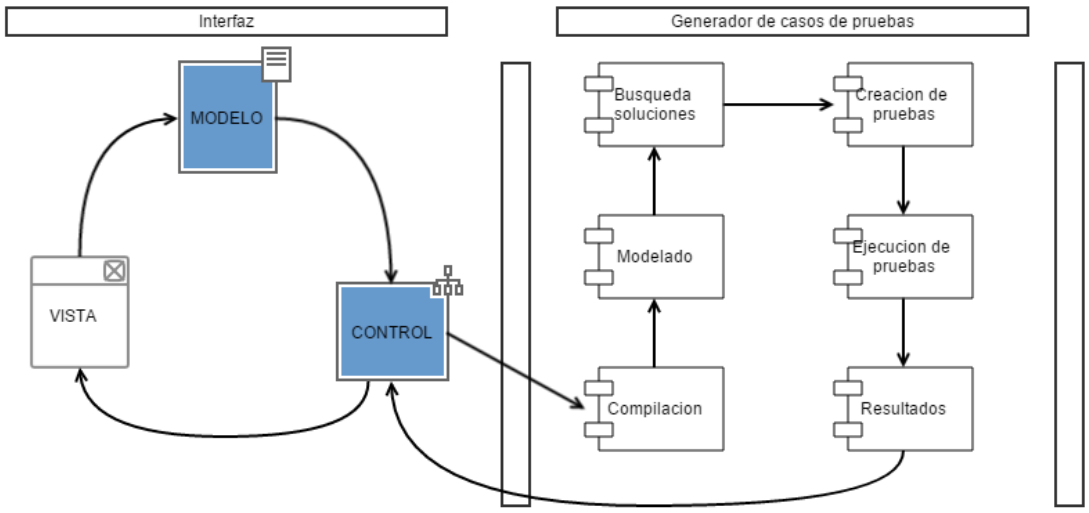



*Figura 4 Diagrama de algoritmo de generador*

También podemos apreciar el flujo en un diagrama de secuencia de UML

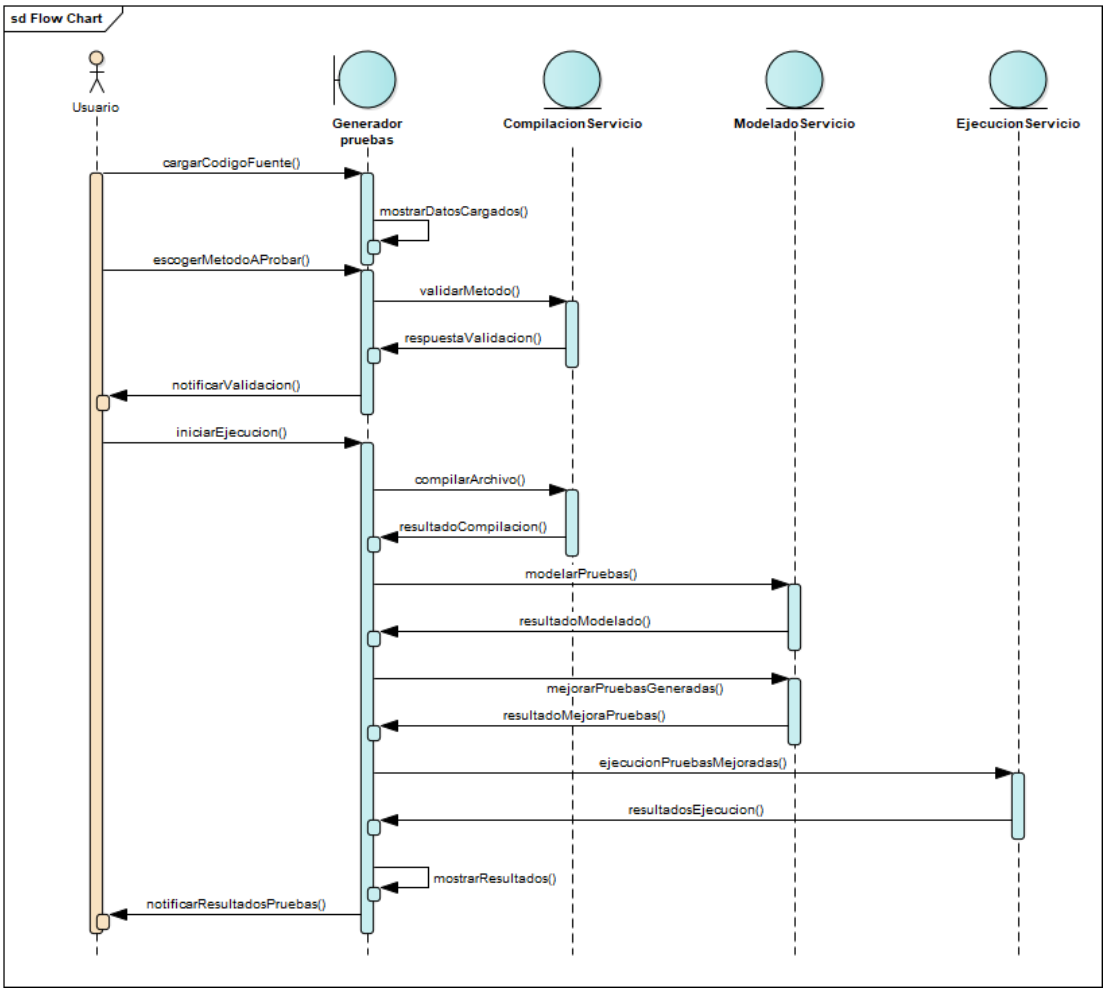



*Figura 5 Diagrama de secuencia para la ejecución de la herramienta*

Y el diagrama de clases UML asociado a este proceso.

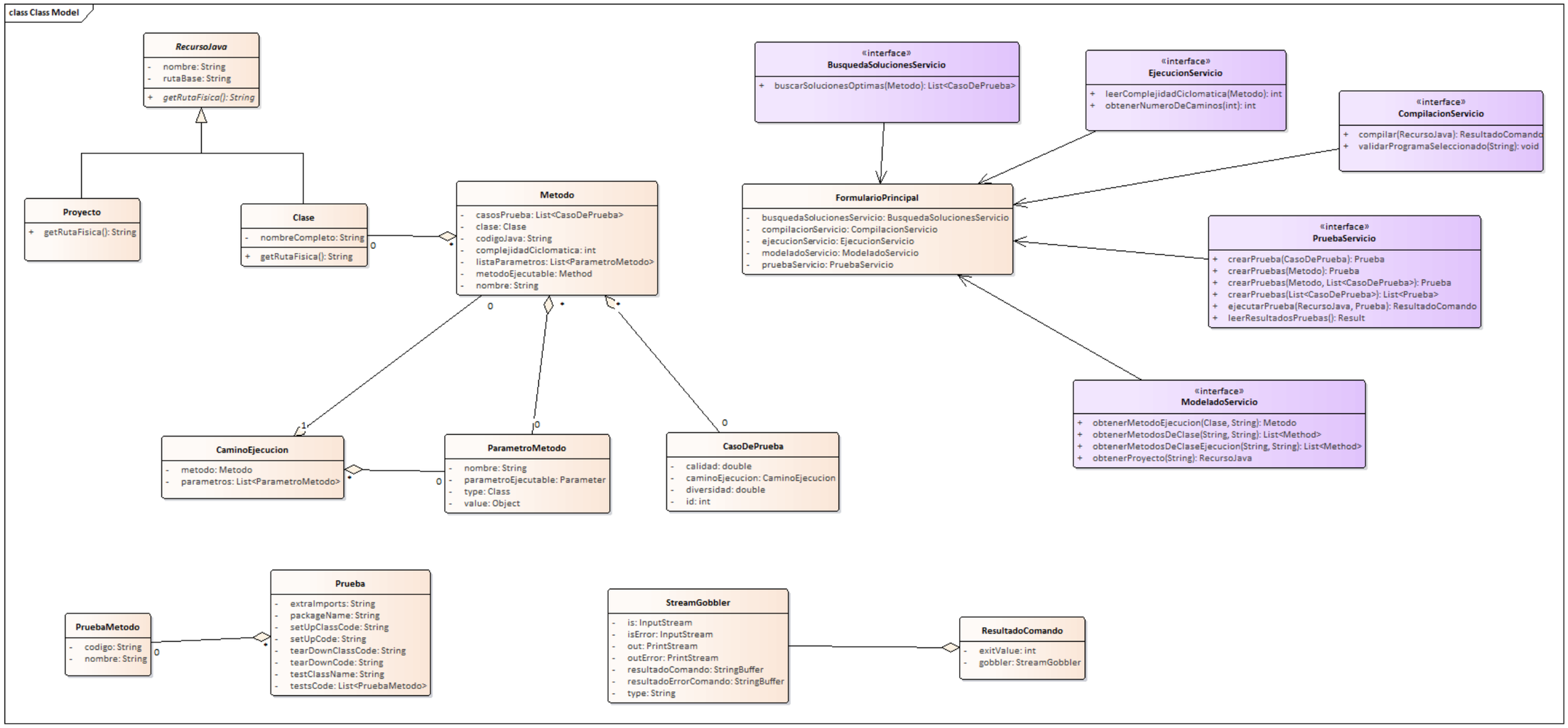



*Figura 6 Diagrama de clases de la herramienta de automatización*

### 3.1.6. Consideraciones de seguridad durante el desarrollo

Cuando se desarrolla algún tipo de software sin importar cuál sea la finalidad que vaya a tener el mismo, es importante que se tomen algunos recaudos, ya que, al tratarse de un sistema tan delicado, es importante que tengamos en cuenta que en cualquier momento puede sufrir alguna falla en su funcionamiento.

Generalmente la seguridad en el desarrollo de software es tan importante como cuando se lo está ejecutando ya que lógicamente, para poder desarrollarlo, se requiere de un sistema operativo especial, el cual también está en riesgo.

Entonces, ¿Cuál es el papel de la seguridad en el desarrollo de software? Generalmente evita que se produzcan errores generales en los sistemas operativos que se utilizan tanto para el desarrollo del mismo, como para las pruebas piloto del funcionamiento del software. Al tratarse de un prototipo de prueba, es muy común que se produzcan fallas permanentemente, y que esto pueda afectar al sistema, además, debemos decir que las fallas en el sistema de software que se está desarrollando necesitan ser ejecutadas para poder corregirlas, ya que, de otro modo, no se podría desarrollar ningún tipo de programa que tenga fallas al ejecutarlo. Por eso, teniendo en cuenta que las fallas deben existir, aunque las mismas pongan en riesgo el funcionamiento del sistema en general, se debe siempre contar con algún programa de seguridad para evitar que las mismas produzcan un daño mayor. En el caso del generador de casos de pruebas comenzaremos enumerando los posibles riesgos que conlleva tanto ejecutar y operar esta herramienta.

1. ¿La ejecución afecta el desempeño del sistema operativo sobre el cual se ejecuta?

R: No, el programa se ejecuta sobre la máquina virtual de java y no uso llamados directos al sistema operativo ni a recursos no administrados, por lo que su ejecución es totalmente segura en cualquier computadora.

2. En caso de falla crítica de la herramienta, ¿Qué les pasará a los datos originales ingresados?

R: No sucederá nada con los datos originales pues, el programa para trabajar crea una copia de los mismos.

3. ¿Se puede sustraer datos del generador?

R: Se puede sustraer la información de configuración de usuario que incluye el directorio de trabajo donde están algunas librerías necesarias y el directorio de salida donde está el código fuente de las pruebas generadas.

4. En caso de fallo externo, ¿Qué sucede en el proceso?

R: Como el fallo es externo y no se puede controlar desde el propio programa, este se detiene de manera intempestiva, pero sin que ello afecte ni a los datos originales ni a futuras ejecuciones del mismo.

5. ¿Quién puede ver los datos que se generan?

R: Aquellos usuarios que tengan permiso de lectura sobre el directorio de salida configurado en las propiedades del programa.

6. ¿Qué datos (de los que se proveen) se guardan y en qué manera?

R: Para una ejecución segura, el código fuente se copia a una carpeta temporal intermedia, la cual, una vez terminado el proceso de ejecución de las pruebas, es eliminada.

### 3.1.7. Diseño de técnicas e instrumentos de recolección de información

Para la realización de la presente investigación se utilizó información de fuentes primarias, fuentes obtenidas de primera mano, para realizar la identificación de los requerimientos necesarios en la herramienta a implementar y para lo cual se utilizaron instrumentos como:

- Observación
- Encuesta

Para el procesamiento, cuantificación, tabulación, diagramas y gráficos de los datos obtenidos se usó un software de hoja de cálculo y debido a la necesidad, otra herramienta de software estadístico más especializada, la cual fue SPSS.

Para complementar la investigación también se usaron fuentes secundarias, como libros, tesis y revistas científicas acordes a los conocimientos informáticos requeridos para realizar la construcción de la herramienta informática propuesta.

### 3.1.8. Materiales y Herramientas

Para la presente investigación se utilizaron los siguientes materiales, recursos y herramientas:

- Laptop 4GB de RAM, procesador Intel I-3 2.20 GHz.
- Sistema Operativo: Debian 8 Jessie 64 bits

- Netbeans IDE 8.2.1

- Compendio bibliográfico, tanto físico como virtual.

### 3.1.9. Operacionalización de variables. Indicadores y/o parámetros

*Tabla 3* Operacionalización de variables

| Variable | Descripción | Indicadores | Indicadores |
|---|---|---|---|
| Generador de casos de prueba para procedimientos en lenguaje java. | Se utilizará en el proyecto actual una herramienta que aplique búsqueda dispersa para encontrar y ejecutar aquellos caminos básicos que cubran la ejecución de una prueban unitaria en su totalidad. | Eficiencia. | Volumen de recursos utilizados bajo ciertas condiciones preestablecidas. |
| | | Tiempo de Respuesta. | Tiempo de demora entre la solicitud y la respuesta |
| | | Eficacia. | Acierto en el proceso conforme a lo que esperaba el usuario. |
| Errores encontrados. | Errores encontrados al ejecutar una prueba unitaria. | Numero de fallas encontradas. | Suma de todos las fallas encontradas que nos brinda la herramienta. |
| | | Presencia del sistema. | Evalúa si la herramienta se está utilizando o no. |

## 3.2. Cobertura del Estudio

### 3.2.1. Población y muestra

En el presente trabajo de investigación la población es todo software desarrollado bajo el lenguaje de programación java y del cual se posea el código fuente.

Para la determinación de la muestra se escogió el software clínico de la Clínica Carita Feliz debido a que, como indican las precondiciones, se necesita acceso al código fuente, y este ha sido brindado, aunque de manera limitada y solo de una cierta cantidad de procesos, entre los que destacan los siguientes:

- Determinar del monto de honorario de una admisión.

- Determinar si una atención es gratuita o no.
- Determinar si se aplicaran incrementos de honorarios a una admisión dependiendo del horario de trabajo normal de la clínica.
- Determinar el tipo mime de cualquier archivo.
- Calculo de horarios y citas médicas diarias
- Liquidaciones de atenciones
- Pre facturación de atenciones.
- Obtención del stock y precio de venta de un producto.

### 3.3. Técnicas e Instrumentos para la Recolección de Datos

Para contrastar los resultados entre las fallas recogidas por el generador de casos de pruebas y una búsqueda manual, se ha ideado una forma de recolección para cada uno, para la búsqueda manual, el instrumento que se utilizó fue una encuesta de investigación la cual se divide en dos partes, la primera, centrada en medir las dimensiones de las variables dependientes y; la segunda, centrada en el nivel de actualización que se tiene sobre el tema de pruebas de software. Los ítems de la encuesta se encuentran en el Anexo 1: Encuesta De Investigación.

La encuesta fue aplicada de la siguiente manera:

*Tabla 4* Personas encuestadas

| N° | Responsable | Empresa | Sistema |
|---|---|---|---|
| 1 | Jefe de Desarrollo | Caja Paita | Sistemas varios |
| 2 | Jefe de Área de T.I. | Carita Feliz | Sistema clínico |
| 3 | Scrum Master Agile | Everis Perú S.A.C. | Cliente BCP |
| 4 | Desarrollador | IBM | Sistemas varios |
| 5 | Desarrollador | PetroPerú | Sistemas varios |
| 6 | Desarrollador | Everis Perú S.A.C. | Cliente Interbank |
| 7 | Analista de calidad | Caja Piura | Área Calidad |
| 8 | Desarrollador | EPS Grau | Sistemas varios |

| | | | |
|---|---|---|---|
| 9 | Desarrollador | FreeLancer | Sistemas varios |
| 10 | Desarrollador | Caja Piura | Área Desarrollo |
| 11 | Desarrollador web | Software Good Idea | Sistemas varios |
| 12 | Analista de sistemas | Jasoft Solutions | Sistemas varios |
| 13 | Scrum Master Agile | FreeLancer | Sistemas varios |
| 14 | Desarrollador Web | Obinte | Sistemas varios |
| 15 | Desarrollador | Wander | Sistemas varios |

Mientas que, para recolectar datos del generador, el proceso se realizó tomando los valores que el propio software provee, siendo condición necesaria, poseer acceso al código fuente del programa o la serie de programas a los cuales se quiere evaluar para buscar fallas. La generación sigue el proceso descrito a continuación:

1. Se inicia el programa generador de casos de pruebas.
2. Se selecciona el código fuente de entrada, ya que en este caso el código está organizado en paquetes, se carga una carpeta completa.
3. Se selecciona ejecutar, lo que inicia el proceso algorítmico descrito en el apartado 3.1.5 Construcción de los Backlogs definidos
4. El generador luego de terminar de procesar, mostrará un gráfico de resumen y un botón para ver los resultados.
5. Seleccionando el botón “Ver detalle”, se muestra de forma tabular la ejecución de los casos de pruebas.

Para más detalle, ver el Anexo 2: Flujo de ejecución del Generador de Casos de Pruebas.

## 3.4. Técnicas de Análisis y Procesamiento de Información

En este apartado se describen las distintas operaciones a las que serán sometidos los datos o respuestas que se obtengan: clasificación, registro, tabulación y codificación

según sea el caso. También se definirán las Técnicas Lógicas o Estadísticas, que se emplearán para descifrar lo que revelan los datos recolectados.

Debido a que el estudio va a comparar una medición de la misma variable bajo distintas condiciones, una usando el generador desarrollado y otra recolectando datos de una encuesta, es necesario aplicar una prueba de diferencia de datos, siendo la prueba de medias la escogida debido a su facilidad de aplicación y utilidad.

Para realizar esta prueba, se necesitan tener algunos datos de tendencia central, como la media, que será descrita más adelante, y también se debe cumplir ciertas condiciones, una de ellas, por ejemplo, es que la población sobre la cual se aplica debe ser normal o tener tendencia hacia a una distribución normal, para validar esto último se utilizó una prueba de homogeneidad de varianzas, que a su vez necesita ciertos estadísticos como la varianza y la desviación estándar, también descritas más adelante.

El proceso que se siguió para probar o rechazar la hipótesis del estudio fue; primero, los datos recogidos tanto de la encuesta como del generador debían estar bajo las mismas unidades de medida; por lo que, a las mediciones del generador se les aplicó estadística descriptiva para agruparse en clases con rangos y unidades similares a los medidos en la encuesta, para lo cual se tuvo que promediar con el valor medio del tiempo de pruebas por semana; seguidamente, se realizó una prueba de homogeneidad de varianzas que como se mencionó en el párrafo anterior, sirve para validar que la población sometida al estudio sigue una distribución normal, lo cual es condición necesaria para una prueba de medias; por último se realizó la prueba de medias en sí misma, para averiguar si existe diferencia estadísticamente significativa entre las medidas con el generador de casos de pruebas desarrollado y los datos recogidos de la encuesta.

A continuación, se detallan los instrumentos utilizados:

**a) Para medir las variables de investigación:**

- **Eficiencia:** Volumen de recursos utilizados bajo ciertas condiciones preestablecidas, este será medido en horas/hombre, es decir, cuanto tiempo les toma a cuantas personas realizar la tarea de escribir pruebas.

- **Tiempo de respuesta:** Tiempo de demora entre la solicitud y la respuesta medido en segundos.

- **Eficacia:** Acierto en el proceso conforme a los que esperaba el usuario, solo se mide la presencia de la misma y viene dado por los requisitos funcionales del generador (creación de casos de pruebas, ejecución de pruebas, mostrar código de pruebas generadas).

- **Número de fallas encontradas:** Suma de todas las fallas encontradas que nos brinda la herramienta.

- **Presencia del sistema:** Evalúa si la herramienta se está utilizando o no, este valor es el decisor para hacer las comparaciones necesarias para probar o rechazar la hipótesis de investigación.

**b) Para conocer tendencias en los datos:**

- **La media aritmética o promedio ($\bar{X}$):** Es el estadístico de tendencia central más significativo y corresponde variables de cualquier nivel de medición, pero particularmente a las mediciones de intervalo y de razón, es usada para realizar una prueba de diferencia de medias que acepte o rechace la hipótesis planteada.

$$\bar{X} = \frac{\sum_{i=1}^{n} x_i * f_i}{n}$$

**Dónde:**

.- $\bar{X}$ : media aritmética

.- $X_i$: marca de clase.

.- $f_i$ : frecuencia de clase

.- $n$ : número total de frecuencias

**c) Para conocer qué tanto varían los datos:**

- **Desviación estándar (S):** Es el promedio de las desviaciones o dispersiones de las puntuaciones respecto a la media o promedio, permite medir el grado de homogeneidad o heterogeneidad de los datos de la población objeto de medición. Cuanto mayor sea la dispersión de los datos respecto a la media mayor será la desviación estándar, lo cual significa mayor heterogeneidad entre las mediciones. Es usada para verificar el Criterio de Homocedasticidad necesario para asumir que la población del estudio es normal o se ajusta a una distribución normal. La fórmula para calcular la desviación estándar de una muestra de observaciones de datos es:

$$S = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}}$$

**Dónde:**

.- $\bar{X}$ : media aritmética

.- $X_i$: marca de clase.

.- $n$ : número total de frecuencias

- **La varianza:** Se define como la elevación al cuadrado de la desviación estándar, $S^2$.

**d) Para describir las diferencias entre grupos y variables:**

- **Prueba de diferencia de medias t-students para muestras independientes:** Es una prueba estadística para evaluar hipótesis en torno a una media cuando los tamaños de la muestra n son menores que 30 mediciones para saber si hay diferencia significativa entre la media de la muestra $\bar{X}$ y la media poblacional $\mu$. Esta prueba está indicada en aquellos casos cuando se quiere establecer si la diferencia entre dos medias de una variable para dos grupos de casos, extraídas de dos poblaciones independientes, son significativamente o si una media es mayor o menor que la otra, es la prueba que se usará para validar la hipótesis del estudio, su fórmula viene dada por:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{S_c * \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

**Dónde:**

.- $\bar{X}_1$ : media aritmética de la muestra 1

.- $\bar{X}_2$ : media aritmética de la muestra 2

.- $n_1$ : número total de frecuencias de la muestra 1

.- $n_2$ : número total de frecuencias de la muestra 2

.- $S_c$ : estimador insesgado de la varianza común $\sigma^2$ definido por:

$$S_c = \sqrt{\frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}}$$

**Dónde:**

.- $S_1^2$ : varianza de la muestra 1

.- $S_2^2$ : varianza de la muestra 2

- **Prueba de homogeneidad de varianzas (la prueba de Levene):** Nos va a permitir verificar el CRITERIO DE HOMOCEDASTICIDAD informándonos sobre el segundo requisito para aplicar la comparación de medias mediante la prueba t de Student: la homogeneidad de varianzas. Esto se logra mediante un contraste a través del estadístico F de Snedecor y nos aporta una significación estadística, o valor "p" asociado a la hipótesis nula de que "las varianzas son homogéneas", de modo que:
    - Si p >= 0.05, p es no significativo, Se asume Homogeneidad.
    - Si p < 0.05, p es significativo, No se asume Homogeneidad.

El programa de computador que se utilizó para el procesamiento de datos y realizar las pruebas respectivas fue el SPSS que es un instrumento de análisis multivalente de datos cuantitativos que está diseñado para el manejo de datos estadísticos.

# Capítulo 4:

# Presentación y Discusión de los Resultados

## 4.1. Presentación de los Resultados

En esta sección se muestran los resultados recogidos tanto de la encuesta realizada como de la ejecución del generador, el análisis y tratamiento matemático se mostrará en la siguiente sección.

Para el generador de casos de pruebas, se precisa que tiene como entrada código de parte del sistema clínico de la Clínica Carita Feliz, que fue proveído en su momento y sigue la ejecución tal como se muestra en el Anexo 2: Flujo de ejecución del Generador de Casos de Pruebas. También es necesario resaltar que este código sólo pertenece a la capa de lógica de negocio, la cual es la más importante en todo software, y que debido al ámbito en la que es definida, solo es visible desde el propio sistema, por ello no realiza validaciones a los datos que recibe como entrada, ya que confía en que las capas superiores se habrán encargado de dicha tarea.

A continuación los resultados en base a los indicadores expuestos en la sección 3.1.9 Operacionalización de variables. Indicadores y/o parámetros.

### 4.1.1. Eficiencia

#### a) Para generador:

Para realizar la generación de todos los casos de pruebas, solo es necesario un operador durante el proceso que lo supervise y extraiga los fallos que se pudieran haber detectado. Y, como podemos observar en la siguiente imagen, se crean 629 casos de pruebas en 194.653 segundos.

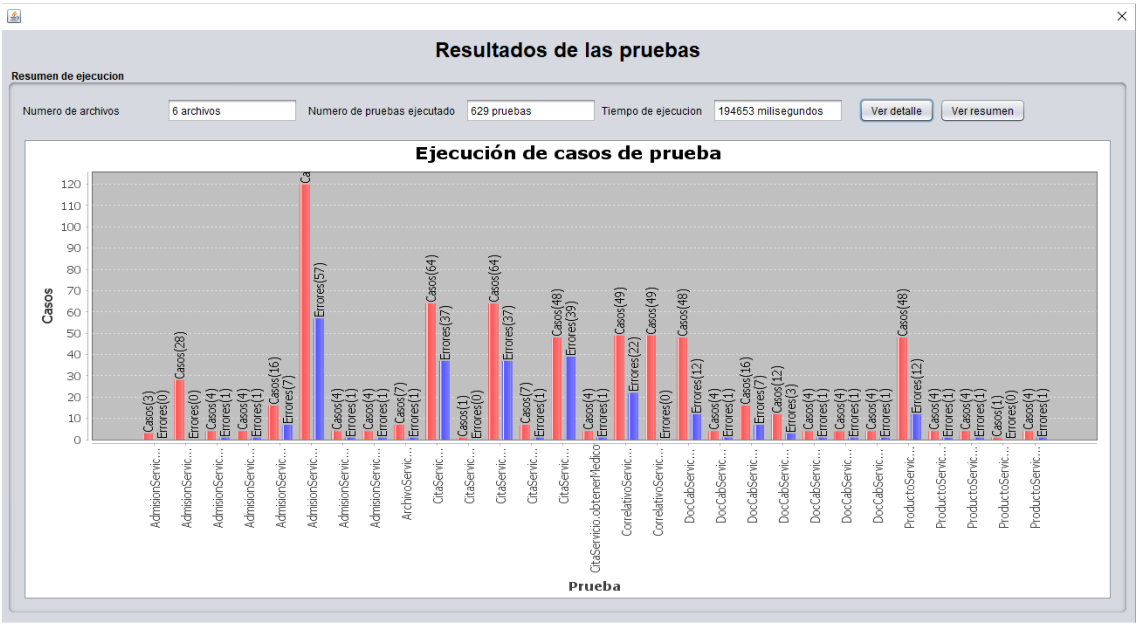



*Figura 7 Resultados de ejecución de pruebas*

## b) Para encuesta:

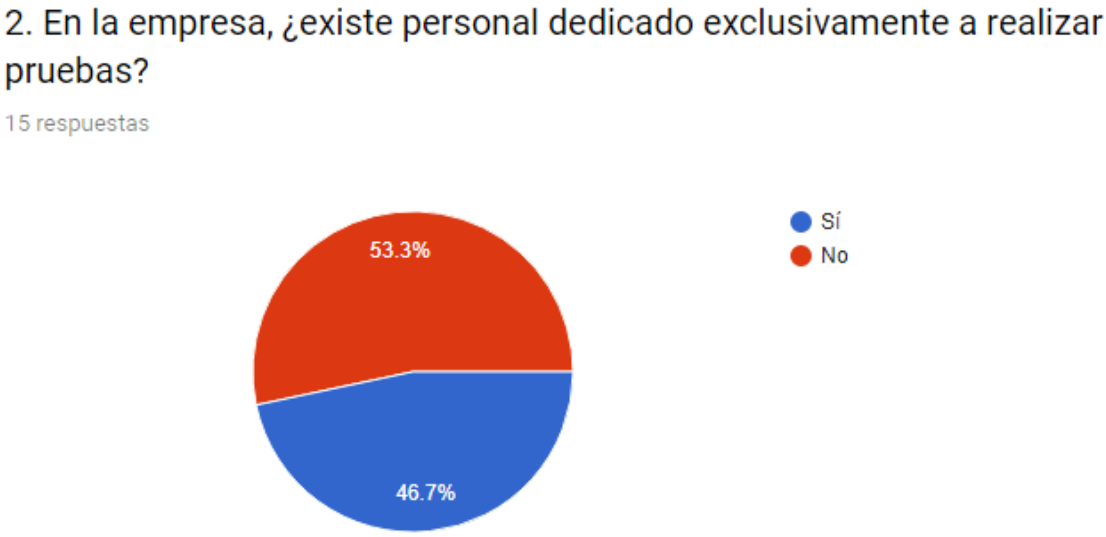



*Figura 8 Gráfico de torta de resultados de la pregunta número 2 de la encuesta de investigación*

### 4.1.2. Tiempo de respuesta

a) Para generador: Como vimos en la sección anterior el tiempo total de ejecución de la generación fue de 194.653 segundos, adicional a ello, la herramienta nos provee el tiempo de ejecución de cada procedimiento y caso de prueba por separado, para de esta manera sea posible analizar las fallas encontradas en cierta cantidad de tiempo.

Detalle de pruebas

| N° | Archivo | Metodo | Pruebas ejecutadas | Errores | Tiempo(ms) | Detalle |
|---|---|---|---|---|---|---|
| 1 | AdmisionServicio | admisionEnHorarioIncremento | 3 | 0 | 31 | Ver Detalles |
| 2 | AdmisionServicio | esAdmisionGratuita | 28 | 0 | 47 | Ver Detalles |
| 3 | AdmisionServicio | tieneTarjetaDescuento | 4 | 1 | 34 | Ver Detalles |
| 4 | AdmisionServicio | obtenerHonorarioMaximo | 4 | 1 | 45 | Ver Detalles |
| 5 | AdmisionServicio | tieneTarifaSocial | 16 | 7 | 40 | Ver Detalles |
| 6 | AdmisionServicio | obtenerHonorarioAdmision | 120 | 57 | 149 | Ver Detalles |
| 7 | AdmisionServicio | obtenerTipoDeAdmision | 4 | 1 | 45 | Ver Detalles |
| 8 | AdmisionServicio | esAsegurado | 4 | 1 | 33 | Ver Detalles |
| 9 | ArchivoServicio | getMimeType | 7 | 1 | 36 | Ver Detalles |
| 10 | CitaServicio | obtenerCitasDeDia | 64 | 37 | 179 | Ver Detalles |
| 11 | CitaServicio | obtenerEstadoCitaNew | 1 | 0 | 51 | Ver Detalles |
| 12 | CitaServicio | obtenerTurnosDeDia | 64 | 37 | 102 | Ver Detalles |
| 13 | CitaServicio | obtenerEstadoCita | 7 | 1 | 75 | Ver Detalles |
| 14 | CitaServicio | obtenerCitasEnHorario | 48 | 39 | 113 | Ver Detalles |
| 15 | CitaServicio | obtenerMedico | 4 | 1 | 70 | Ver Detalles |
| 16 | CorrelativoServicio | siguienteNroCorrelativo | 49 | 22 | 68 | Ver Detalles |
| 17 | CorrelativoServicio | obtenerPorNombreYAnio | 49 | 0 | 114 | Ver Detalles |
| 18 | DocCabServicio | liquidacionEmergencia | 48 | 12 | 432 | Ver Detalles |
| 19 | DocCabServicio | obtenerDocumExamenesDet | 4 | 1 | 62 | Ver Detalles |
| 20 | DocCabServicio | obtenerPrefactura | 16 | 7 | 61 | Ver Detalles |
| 21 | DocCabServicio | liquidarAdmisionEmergencia | 12 | 3 | 44 | Ver Detalles |
| 22 | DocCabServicio | obtenerDocumProductoDet | 4 | 1 | 62 | Ver Detalles |
| 23 | DocCabServicio | obtenerDocumServicioDet | 4 | 1 | 64 | Ver Detalles |
| 24 | DocCabServicio | obtenerLista | 4 | 1 | 120 | Ver Detalles |
| 25 | ProductoServicio | obtenerProductoParaVenta | 48 | 12 | 106 | Ver Detalles |
| 26 | ProductoServicio | obtenerStockDeProducto | 4 | 1 | 46 | Ver Detalles |
| 27 | ProductoServicio | obtenerInclusoEliminado | 4 | 1 | 49 | Ver Detalles |
| 28 | ProductoServicio | obtenerDescuentoFarmacia | 1 | 0 | 42 | Ver Detalles |
| 29 | ProductoServicio | obtenerInformedicoTratamiento | 4 | 1 | 62 | Ver Detalles |

*Figura 9 Detalle de ejecución de pruebas automáticas*

**b) Para encuesta:**


*Figura 10 Gráfico de torta de resultados de la pregunta número 3 de la encuesta de investigación*

### 4.1.3. Eficacia

#### a) Para generador:

Como vemos en la imagen a continuación, el generador cumple con los requisitos funcionales definidos, por lo que este indicador queda cubierto.

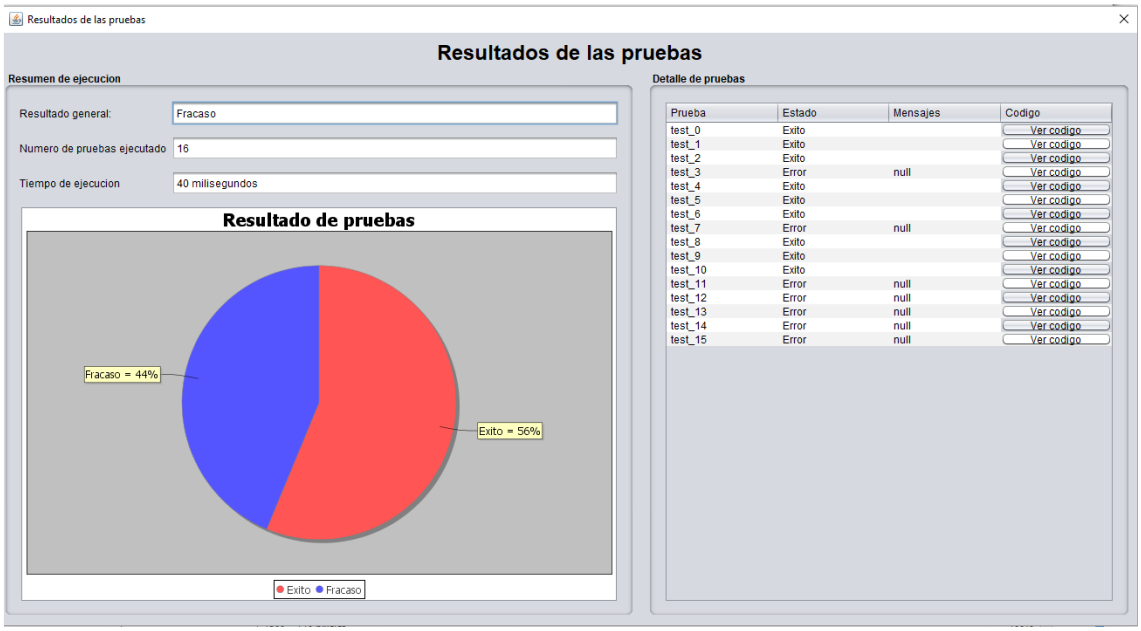



*Figura 11 Resultados de un caso de prueba*


*Figura 12 Código de prueba generado automáticamente*

**b) Para encuesta:**


*Figura 13 Gráfico de torta de resultados de la pregunta número 5 de la encuesta de investigación*

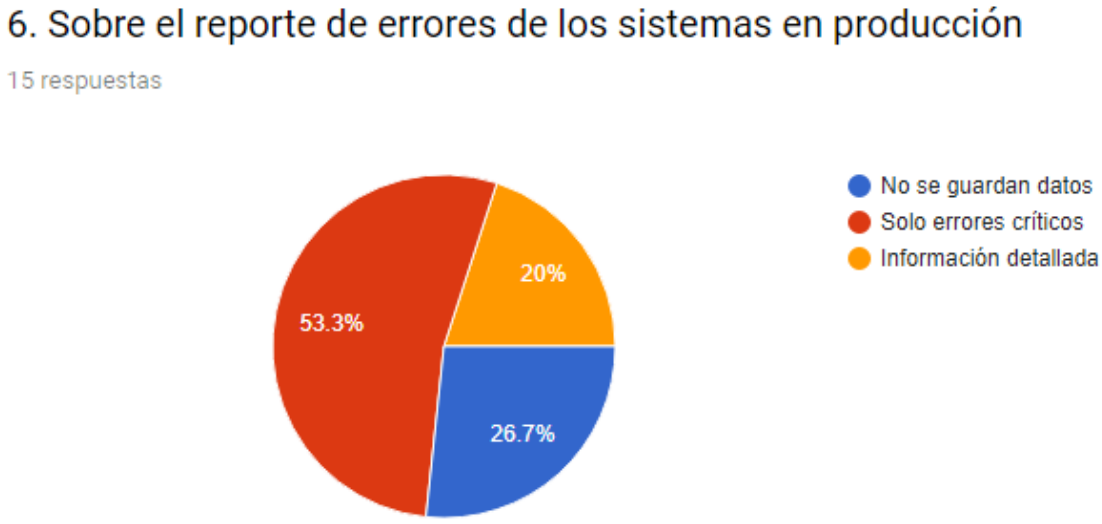



*Figura 14 Gráfico de torta de resultados de la pregunta número 6 de la encuesta de investigación*

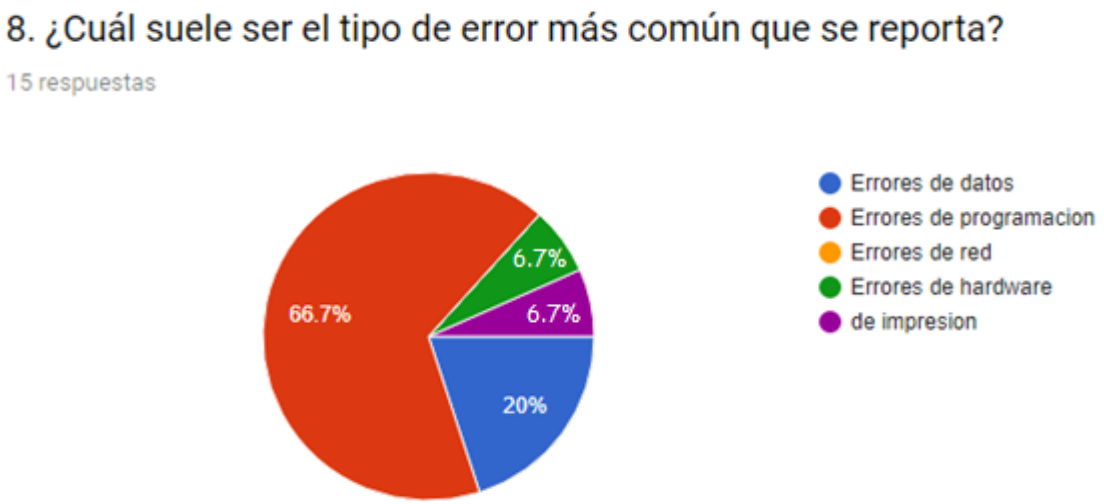


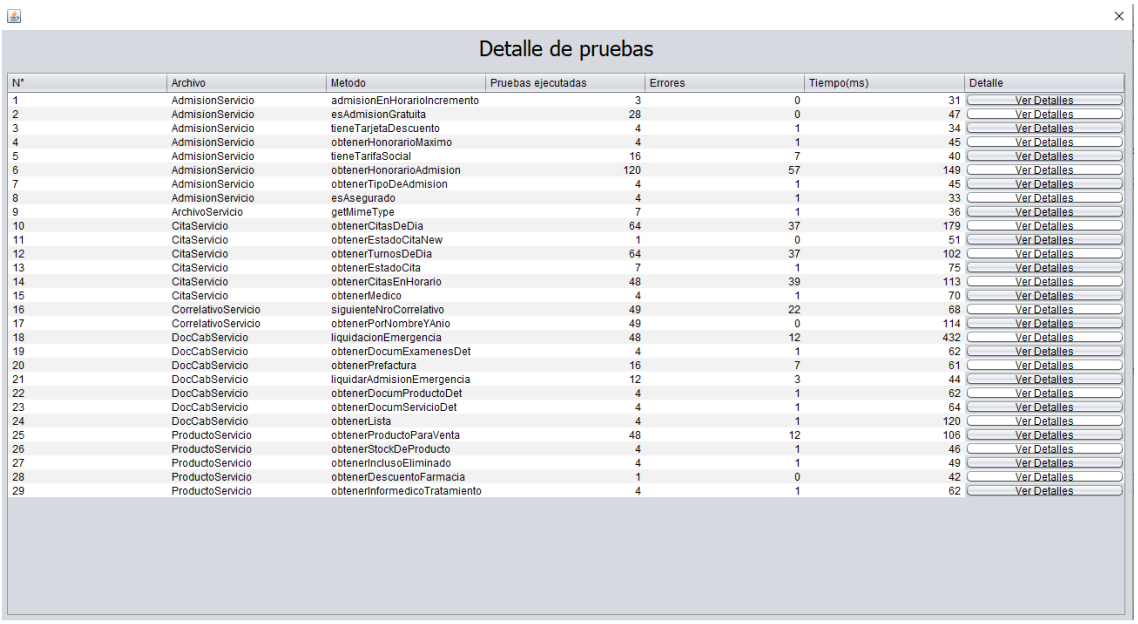

*Figura 15 Gráfico de torta de resultados de la pregunta número 8 de la encuesta de investigación*

### 4.1.4. Número de fallas encontradas

#### a) Para generador:

En este indicador, el generador muestra los resultados de manera agrupada o detallada, tal como se muestra a continuación:


*Figura 16 Fallas encontradas - resumen*




| N° | Archivo | Metodo | Pruebas ejecutadas | Errores | Tiempo(ms) | Detalle |
|---|---|---|---|---|---|---|
| 1 | AdmisionServicio | admisionEnHorarioIncremento | 3 | 0 | 31 | Ver Detalles |
| 2 | AdmisionServicio | esAdmisionGratuita | 28 | 0 | 47 | Ver Detalles |
| 3 | AdmisionServicio | tieneTarjetaDescuento | 4 | 1 | 34 | Ver Detalles |
| 4 | AdmisionServicio | obtenerHonorarioMaximo | 4 | 1 | 45 | Ver Detalles |
| 5 | AdmisionServicio | tieneTarifaSocial | 16 | 7 | 40 | Ver Detalles |
| 6 | AdmisionServicio | obtenerHonorarioAdmision | 120 | 57 | 149 | Ver Detalles |
| 7 | AdmisionServicio | obtenerTipoDeAdmision | 4 | 1 | 45 | Ver Detalles |
| 8 | AdmisionServicio | esAsegurado | 4 | 1 | 33 | Ver Detalles |
| 9 | ArchivoServicio | getMimeType | 7 | 1 | 36 | Ver Detalles |
| 10 | CitaServicio | obtenerCitasDeDia | 64 | 37 | 179 | Ver Detalles |
| 11 | CitaServicio | obtenerEstadoCitaNew | 1 | 0 | 51 | Ver Detalles |
| 12 | CitaServicio | obtenerTurnosDeDia | 64 | 37 | 102 | Ver Detalles |
| 13 | CitaServicio | obtenerEstadoCita | 7 | 1 | 75 | Ver Detalles |
| 14 | CitaServicio | obtenerCitasEnHorario | 48 | 39 | 113 | Ver Detalles |
| 15 | CitaServicio | obtenerMedico | 4 | 1 | 70 | Ver Detalles |
| 16 | CorrelativoServicio | siguienteNroCorrelativo | 49 | 22 | 68 | Ver Detalles |
| 17 | CorrelativoServicio | obtenerPorNombreYAnio | 49 | 0 | 114 | Ver Detalles |
| 18 | DocCabServicio | liquidacionEmergencia | 48 | 12 | 432 | Ver Detalles |
| 19 | DocCabServicio | obtenerDocumExamenesDet | 4 | 1 | 62 | Ver Detalles |
| 20 | DocCabServicio | obtenerPrefactura | 16 | 7 | 61 | Ver Detalles |
| 21 | DocCabServicio | liquidarAdmisionEmergencia | 12 | 3 | 44 | Ver Detalles |
| 22 | DocCabServicio | obtenerDocumProductoDet | 4 | 1 | 62 | Ver Detalles |
| 23 | DocCabServicio | obtenerDocumServicioDet | 4 | 1 | 64 | Ver Detalles |
| 24 | DocCabServicio | obtenerLista | 4 | 1 | 120 | Ver Detalles |
| 25 | ProductoServicio | obtenerProductoParaVenta | 48 | 12 | 106 | Ver Detalles |
| 26 | ProductoServicio | obtenerStockDeProducto | 4 | 1 | 46 | Ver Detalles |
| 27 | ProductoServicio | obtenerInclusoEliminado | 4 | 1 | 49 | Ver Detalles |
| 28 | ProductoServicio | obtenerDescuentoFarmacia | 1 | 0 | 42 | Ver Detalles |
| 29 | ProductoServicio | obtenerInformedicoTratamiento | 4 | 1 | 62 | Ver Detalles |

*Figura 17 Fallas encontradas – detalle*

**b) Para encuesta:**

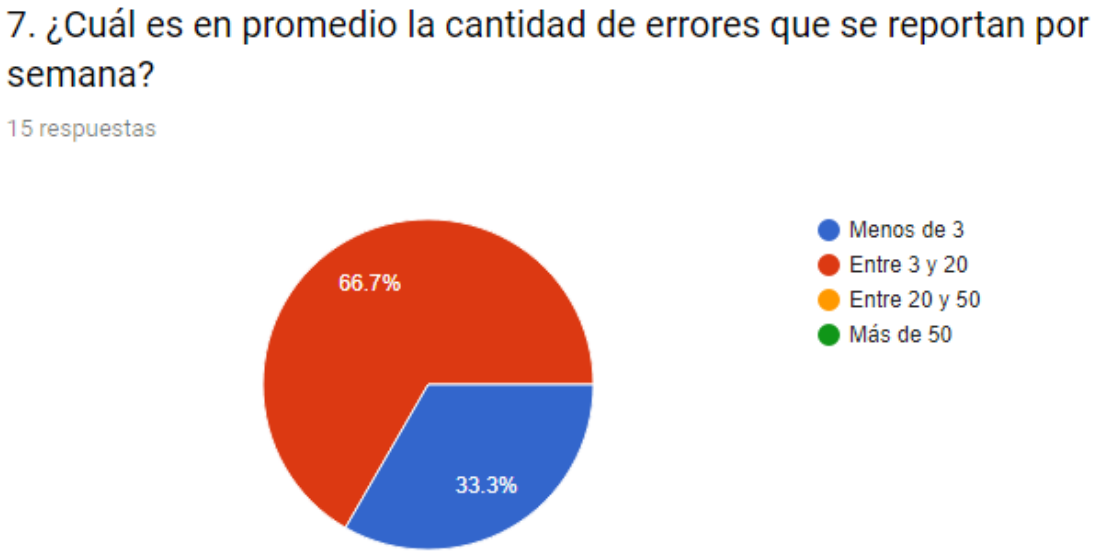



*Figura 18 Gráfico de torta de resultados de la pregunta número 7 de la encuesta de investigación*

### 4.1.5. Otros resultados

Adicional se muestra el resultado de las respuestas de la encuesta para el resto de preguntas, las cuales fueron necesarias para saber el nivel de manejo sobre el tema que tenían los encuestados, y para poder brindar recomendaciones sobre actualizaciones sobre el cambiante ambiente de pruebas de software y automatización.


*Figura 19 Gráfico de torta de resultados de la pregunta número 1 de la encuesta de investigación*

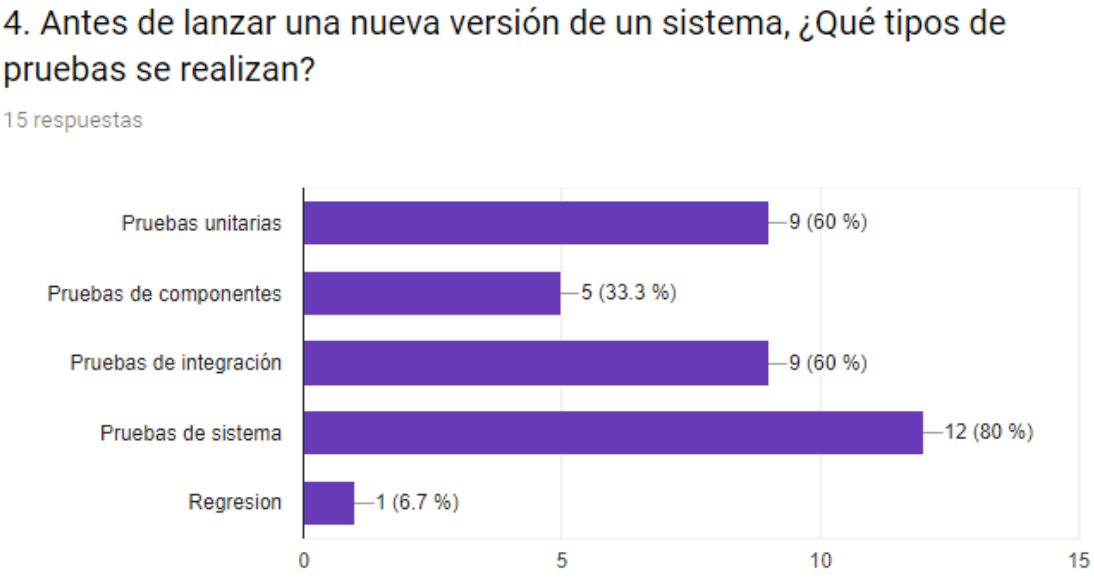



*Figura 20 Gráfico de torta de resultados de la pregunta número 4 de la encuesta de investigación*

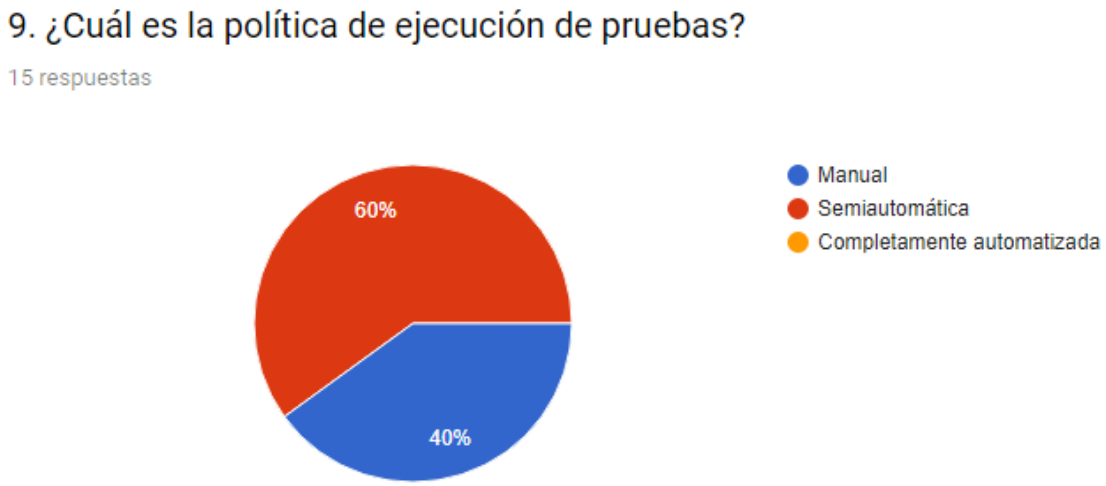



*Figura 21 Gráfico de torta de resultados de la pregunta número 9 de la encuesta de investigación*

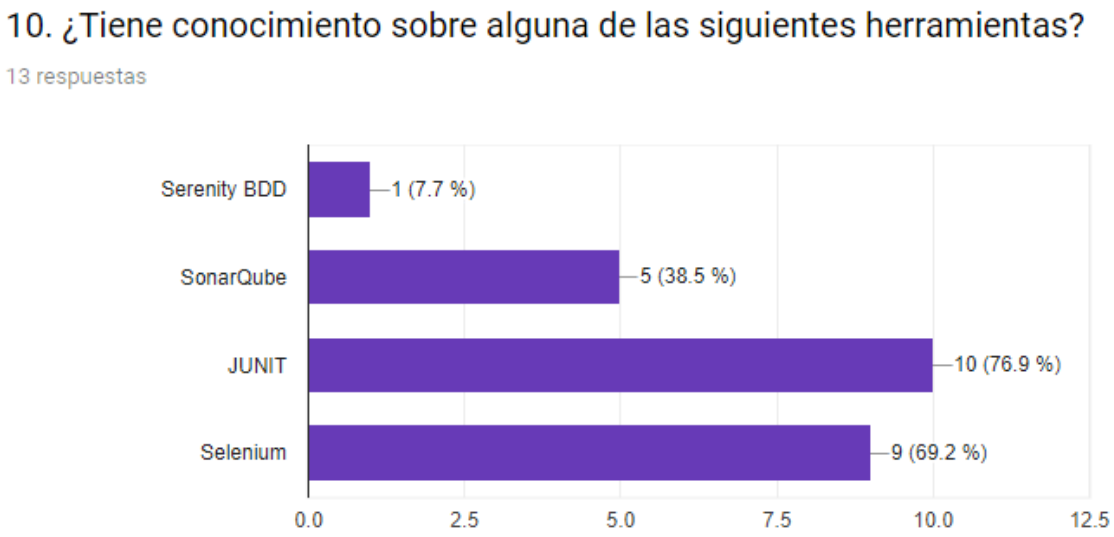



*Figura 22 Gráfico de torta de resultados de la pregunta número 10 de la encuesta de investigación*

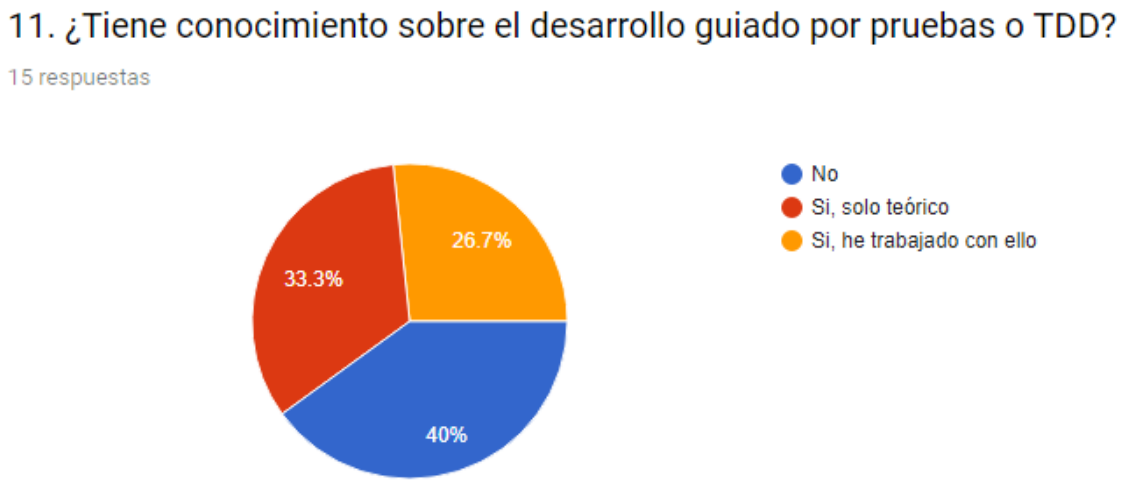



*Figura 23 Gráfico de torta de resultados de la pregunta número 11 de la encuesta de investigación*

## 4.2. Análisis de los Resultados

Para la verificación de la hipótesis se usaron los datos de la encuesta y los datos recogidos del generador, los cuales se muestran a continuación:

De la encuesta realizada, la pregunta número 3 indica lo siguiente:

¿Cuál es el tiempo a la semana que se dedica a realizar pruebas?

a. Menos de 10 horas
b. Entre 10 y 20 horas
c. Entre 20 y 30 horas
d. Más de 30 horas

La cual obtuvo los siguientes resultados:

*Tabla 5 Resultados de la pregunta número 3 de la encuesta de investigación.*

| N° | Fallas | Ocurrencias |
|---|---|---|
| 1 | De 0 a 10 horas | 9 |
| 2 | De 10 a 20 horas | 0 |
| 3 | De 20 a 30 horas | 1 |
| 4 | Más de 30 horas | 5 |
| **Total** | | 15 |

La pregunta número 7 indica lo siguiente:

¿Cuál es en promedio la cantidad de errores que se reportan por semana?

a. Menos de 3

b. Entre 3 y 20

c. Entre 20 y 50

d. Más de 50

La cual, obtuvo los siguientes resultados:

*Tabla 6* Resultados de la pregunta número 7 de la encuesta de investigación.

| N° | Fallas | Ocurrencias |
|---|---|---|
| 1 | De 0 a 3 | 5 |
| 2 | De 3 a 20 | 10 |
| 3 | De 20 a 50 | 0 |
| 4 | Más de 50 | 0 |
| **Total** | | 15 |

Dado que, es necesario tratar estadísticamente los datos obtenidos, estos se agruparán en clases para facilitar el proceso de comprobación de hipótesis.

*Tabla 7* Tiempo promedio gastado en pruebas de software según encuesta

| N° | Tiempo | Marca de clase | Ocurrencias |
|---|---|---|---|
| 1 | De 0 a 10 horas | 5.00 | 9 |
| 2 | De 10 a 20 horas | 15.00 | 0 |
| 3 | De 20 a 30 horas | 25.00 | 1 |
| 4 | De 30 a 40 horas | 35.00 | 5 |
| **Total** | | | 15 |

El primer paso a realizar será promediar la cantidad de horas a la semana que se realizan las pruebas, esto se hará sacando el promedio a los datos obtenidos.

$$\bar{X} = 16.33\ horas$$

Expresando el promedio en segundos, que será la medida en la que se trabajará el tiempo en adelante se tiene:

$$\bar{X} = 58{,}800\ s$$

Y continuamos con la cantidad de errores.

*Tabla 8* Numero de fallas reportadas de los sistemas informáticos en producción según encuesta.

| N° | Fallas por semana | Ocurrencias |
|---|---|---|
| 1 | De 0 a 3 | 5 |
| 2 | De 3 a 20 | 10 |
| 3 | De 20 a 50 | 0 |
| 4 | De 50 a 100 | 0 |
| **Total** | | 15 |

Ahora se calculará la variable de investigación para los datos obtenidos, de la siguiente manera:

1 error/semana = 1 error / 58800 s = 0.000017 errores/s

Con lo que el cuadro quedaría de la siguiente manera:

*Tabla 9* Calidad medida usando encuesta de investigación.

| N° | Límite inferior | Límite superior | Marca de clase | Ocurrencias |
|---|---|---|---|---|
| 1 | 0.0000000 | 0.0000510 | 0.0000255 | 5 |
| 2 | 0.0000510 | 0.0003401 | 0.0001956 | 10 |
| 3 | 0.0003401 | 0.0008503 | 0.0005952 | 0 |
| 4 | 0.0008503 | 500.0000000 | 250.0004252 | 0 |
| **Total** | | | | 15 |

Ahora bien, del generador de pruebas construido, se obtienen los siguientes resultados:

*Tabla 10* Numero de errores encontrados con el generador desarrollado.

| N | Archivo | Método/procedimiento | Pruebas | Fallas | Tiempo (s) |
|---|---|---|---|---|---|
| 1 | AdmisionServicio | obtenerHonorarioAdmision | 120 | 57 | 0.1250 |
| 2 | AdmisionServicio | esAsegurado | 4 | 1 | 0.0330 |
| 3 | AdmisionServicio | esAdmisionGratuita | 28 | 0 | 0.0680 |
| 4 | AdmisionServicio | tieneTarjetaDescuento | 4 | 1 | 0.0380 |
| 5 | AdmisionServicio | obtenerTipoDeAdmision | 4 | 1 | 0.0430 |
| 6 | AdmisionServicio | tieneTarifaSocial | 16 | 7 | 0.0400 |
| 7 | AdmisionServicio | admisionEnHorarioIncremento | 3 | 0 | 0.0440 |
| 8 | AdmisionServicio | obtenerHonorarioMaximo | 4 | 1 | 0.0470 |
| 9 | ArchivoServicio | getMimeType | 7 | 1 | 0.0380 |
| 10 | CitaServicio | obtenerMedico | 4 | 1 | 0.0680 |
| 11 | CitaServicio | obtenerCitasDeDia | 64 | 37 | 0.1770 |
| 12 | CitaServicio | obtenerEstadoCitaNew | 1 | 0 | 0.0550 |
| 13 | CitaServicio | obtenerEstadoCita | 7 | 1 | 0.0590 |
| 14 | CitaServicio | obtenerCitasEnHorario | 48 | 39 | 0.1030 |
| 15 | CitaServicio | obtenerTurnosDeDia | 64 | 37 | 0.1450 |
| 16 | CorrelativoServicio | siguienteNroCorrelativo | 49 | 30 | 0.0710 |
| 17 | CorrelativoServicio | obtenerPorNombreYAnio | 49 | 0 | 0.0960 |
| 18 | DocCabServicio | liquidacionEmergencia | 48 | 12 | 0.7700 |
| 19 | DocCabServicio | obtenerPrefactura | 16 | 7 | 0.0680 |
| 20 | DocCabServicio | liquidarAdmisionEmergencia | 12 | 3 | 0.0580 |
| 21 | DocCabServicio | obtenerDocumProductoDet | 4 | 1 | 0.0700 |
| 22 | DocCabServicio | obtenerDocumExamenesDet | 4 | 1 | 0.0710 |
| 23 | DocCabServicio | obtenerDocumServicioDet | 4 | 1 | 0.0630 |
| 24 | DocCabServicio | obtenerLista | 4 | 1 | 0.2710 |
| 25 | ProductoServicio | obtenerProductoParaVenta | 48 | 12 | 0.0880 |
| 26 | ProductoServicio | obtenerInclusoEliminado | 4 | 1 | 0.0490 |
| 27 | ProductoServicio | obtenerInformedicoTratamiento | 4 | 1 | 0.0490 |
| 28 | ProductoServicio | obtenerDescuentoFarmacia | 1 | 0 | 0.0470 |
| 29 | ProductoServicio | obtenerStockDeProducto | 4 | 1 | 0.0600 |
| **Total** | | | **637** | **257** | **2.9140** |

Necesitamos calcular los resultados en la misma medida que la encuesta, las cuales son número de errores por cada segundo, con lo cual se obtiene lo siguiente:

*Tabla 11* Fallas medidas usando el generador de casos de pruebas

| N | Archivo | Método/procedimiento | Fallas/segundo |
|---|---|---|---|
| 1 | AdmisionServicio | obtenerHonorarioAdmision | 456.000000 |
| 2 | AdmisionServicio | esAsegurado | 30.303030 |
| 3 | AdmisionServicio | esAdmisionGratuita | 0.000000 |

| | | | |
|---|---|---|---|
| 4 | AdmisionServicio | tieneTarjetaDescuento | 26.315789 |
| 5 | AdmisionServicio | obtenerTipoDeAdmision | 23.255814 |
| 6 | AdmisionServicio | tieneTarifaSocial | 175.000000 |
| 7 | AdmisionServicio | admisionEnHorarioIncremento | 0.000000 |
| 8 | AdmisionServicio | obtenerHonorarioMaximo | 21.276596 |
| 9 | ArchivoServicio | getMimeType | 26.315789 |
| 10 | CitaServicio | obtenerMedico | 14.705882 |
| 11 | CitaServicio | obtenerCitasDeDia | 209.039548 |
| 12 | CitaServicio | obtenerEstadoCitaNew | 0.000000 |
| 13 | CitaServicio | obtenerEstadoCita | 16.949153 |
| 14 | CitaServicio | obtenerCitasEnHorario | 378.640777 |
| 15 | CitaServicio | obtenerTurnosDeDia | 255.172414 |
| 16 | CorrelativoServicio | siguienteNroCorrelativo | 422.535211 |
| 17 | CorrelativoServicio | obtenerPorNombreYAnio | 0.000000 |
| 18 | DocCabServicio | liquidacionEmergencia | 15.584416 |
| 19 | DocCabServicio | obtenerPrefactura | 102.941176 |
| 20 | DocCabServicio | liquidarAdmisionEmergencia | 51.724138 |
| 21 | DocCabServicio | obtenerDocumProductoDet | 14.285714 |
| 22 | DocCabServicio | obtenerDocumExamenesDet | 14.084507 |
| 23 | DocCabServicio | obtenerDocumServicioDet | 15.873016 |
| 24 | DocCabServicio | obtenerLista | 3.690037 |
| 25 | ProductoServicio | obtenerProductoParaVenta | 136.363636 |
| 26 | ProductoServicio | obtenerInclusoEliminado | 20.408163 |
| 27 | ProductoServicio | obtenerInformedicoTratamiento | 20.408163 |
| 28 | ProductoServicio | obtenerDescuentoFarmacia | 0.000000 |
| 29 | ProductoServicio | obtenerStockDeProducto | 16.666667 |
| **Total** | | | **2467.539637** |

De la misma manera que a los resultados de la encuesta, necesitamos agrupar los datos que obtenemos del generador, lo que nos lleva al siguiente cuadro.

*Tabla 12* Fallas medidas agrupadas en clases.

| N° | Límite inferior | Límite superior | Marca de clase | Ocurrencias |
|---|---|---|---|---|
| 1 | 0.0000000 | 0.0000510 | 0.0000255 | 5 |
| 2 | 0.0000510 | 0.0003401 | 0.0001956 | 0 |
| 3 | 0.0003401 | 0.0008503 | 0.0005952 | 0 |
| 4 | 0.0008503 | 500.0000000 | 250.0004252 | 24 |
| **Total** | | | | 29 |

Sintetizando los datos en un solo cuadro se tiene lo siguiente:

*Tabla 13* Cuadro resumen de datos de errores

| N° | Límite inferior | Límite superior | Marca de clase | Sin generador | Con generador |
|---|---|---|---|---|---|
| 1 | 0.0000000 | 0.0000510 | 0.0000255 | 5 | 5 |
| 2 | 0.0000510 | 0.0003401 | 0.0001956 | 10 | 0 |
| 3 | 0.0003401 | 0.0008503 | 0.0005952 | 0 | 0 |
| 4 | 0.0008503 | 500.0000000 | 250.0004252 | 0 | 24 |
| **Total** | | | | **15** | **29** |

Se requiere comprobar que existe una diferencia significativa entre los puntajes obtenidos por ambos grupos de datos. Como se puede observar en la **Tabla 13** ***Cuadro resumen de datos de errores,*** tenemos los valores de la variable cuantitativa "Calidad de los procedimientos implementados en lenguaje java" y como variable cualitativa categórica "Uso del Generador" con dos estados "Ocurrencias sin generador" y "Ocurrencias con generador" que es la que define los dos grupos de datos.

Como los datos observaciones son menores a 30 unidades de análisis, se utilizó la prueba de T de Student cuyo diseño de procesamiento es el siguiente:

$\boldsymbol{H_0}$: No existe diferencia significativa entre los errores encontrados por el generador de casos de prueba y los errores encontrados mediante plan de pruebas manual.

$\boldsymbol{H_1}$: El generador de casos de prueba encuentra más errores que un plan de pruebas ejecutado de manera manual

O lo que es lo mismo:

$$\boldsymbol{H_0}: \mu SIN = \mu CON$$

$$\boldsymbol{H_1}: \mu SIN < \mu CON$$

**Dónde:**

.- $H_0$ : Hipótesis nula

.- $H_1$ : Hipótesis alterna

.- $\mu SIN$ : Media de errores con generador

.- $\mu CON$ : Media de errores sin generador

**Prueba T de Student:**

*Tabla 14* Estadísticos de grupo

| Variable dependiente | Uso de generador | N | Media | Desviación estándar | Error estándar de la media |
|---|---|---|---|---|---|
| Número de errores | Sin generador | 15 | 0.000138900 | 0.0000830003 | 0.0000214306 |
| | Con generador | 29 | 206.896908000 | 96.1066217100 | 17.8465516200 |

Primero se muestran los estadísticos resumen en cada grupo: N (tamaño), media, desviación estándar y el error estándar de la media.

Luego el programa SPSS nos aporta información de la prueba T de Student en un único cuadro resumen, donde se nos ofrecen varias cosas:

**Una prueba de homogeneidad de varianzas (la prueba de Levene)**, que nos va a informar sobre el segundo requisito para aplicar la comparación de medias mediante la prueba T de Student: la homogeneidad de varianzas. El programa hace un contraste a través del estadístico F de Snedecor y nos aporta una significación estadística, o valor "p" asociado a la hipótesis nula de que "las varianzas son homogéneas". Cuando ese valor "p" es significativo ($p<0,05$) debemos dudar de la homogeneidad de varianzas.

*Tabla 15* Prueba de muestras independientes T-Student de la variable número de errores encontrados

| **Calidad de software** | **Prueba de Levene para la igualdad de varianzas** | | **Prueba T para la igualdad de medias** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **F** | **Sig.** | **t** | **gl** | **Sig(bilateral)** | **Diferencia de medias** | **Error típico de la diferencia** | **95% intervalo de confianza para la diferencia** | |
| | | | | | | | | **Superior** | **Inferior** |
| Se han asumido varianzas iguales | 19.038 | 0.000082 | -8.290 | 42 | 0.000 | -206.8967691 | 24.95682833 | -257.2616877 | -156,5318505 |
| No se han asumido varianzas iguales | | | -11.593 | 28.000 | 0.000 | -206.8967691 | 17.84655162 | -243,4537729 | -170,3397653 |

## 4.3. Interpretación de los Resultados

Como se puede observar en el cuadro anterior la prueba de Levene es significativa (p =0.000082), por lo que asumimos la heterogeneidad de varianzas y leemos la T de Student en la fila inferior (“No se han asumido varianzas iguales”): el estadístico t vale -11.593 (con 28.000 grados de libertad).

**Entonces tenemos que:**

- A un nivel de significancia alfa de 5% → α=0.05
- Grados de libertad → gl=28.000
- T de Student calculado → Tc= -11.593
- T de Student de tabla → Tt= -1,6896

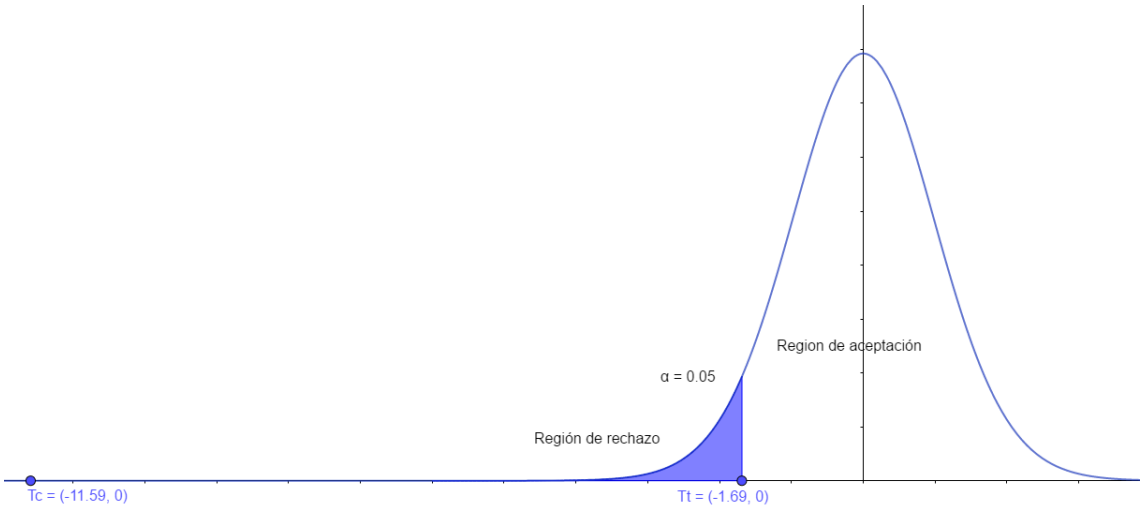



*Figura 24 Gráfica de resultados de hipótesis*

**Partiendo de los considerandos:**

- Rechazo la hipótesis Nula si: Tc > Tt ó –Tc < -Tt
- Acepto la hipótesis Nula si: Tc <=Tt ó –Tc >= -Tt

Entonces como -Tc < -Tt (-11.593 < -1.6896), se rechaza la hipótesis nula. Este hecho nos permite afirmar de forma innegable que "**El uso del generador de casos de prueba para procedimientos en lenguaje java encuentra más fallas que el método tradicional de buscar fallas manualmente**", al haberse encontrado una diferencia estadísticamente significativa entre ambos grupos.

### 4.4. Discusión de los Resultados

Las pruebas de software son una parte fundamental del ciclo de vida de un proyecto, y conseguir crear un generador de casos de pruebas automático que se encargue de dicha tarea por nosotros puede ser un gran avance en la automatización y mejora de los sistemas que actualmente se desarrollan, durante la investigación se han encontrado pruebas irrefutables que demuestran que, la herramienta construida en el presente trabajo, y en consecuencia, otras basadas en las mismas técnicas, obtienen mejores resultados que un plan de pruebas manual, es decir, se encuentran más fallas, por lo que la calidad de software mejora usando el generador desarrollado, asimismo, cabe resaltar que el alcance de este estudio sólo es la generación de los casos de pruebas, la subsiguiente subsanación de las fallas no es cubierta y bien podría introducir nuevas fallas, pero que pueden ser analizadas de manera incremental con esta herramienta, también debo destacar que actualmente hay nuevos enfoques de desarrollo de software basado en pruebas primero, como lo son el TDD (*Test-driven development* o desarrollo guiado por pruebas en español) o el BDD (*Behavior-driven development* o desarrollado guiado por comportamiento) que podrían en el futuro requerir un nuevo estudio para comprobar si una herramienta de automatización es necesaria en fase de desarrollo.

La culminación de este trabajo deja un campo futuro para continuar su desarrollo, o bien para ser tomado como base en investigaciones en las que se pretenda mejorar el proceso de aseguramiento de la calidad de un software mediante pruebas unitarias, debido

a que provee un marco que demuestra experimentalmente que generar casos de prueba de manera automática usando técnicas de pruebas de caja blanca y búsqueda dispersa provee una mejora significativa; o bien, se puede probar usando otras técnicas existentes en su reemplazo que pueden generar resultados más favorables, como lo son las pruebas de caja negra, búsqueda tabú o búsqueda local.

## Conclusiones

1. Se determinó que la implementación de un generador de casos de prueba para procedimientos en lenguaje java influyó en la mejora de la calidad del software estudiado porque se diseñó teniendo una visión detallada y explicativa de las pruebas unitarias, su utilidad y la ventaja de automatizar su creación.

2. Los errores de programación constituyeron el 66.7% de los fallos reportados según la encuesta realizada, concluyendo que en fase de desarrollo no se llegan a realizar suficientes pruebas para aumentar la calidad de un software.

3. La implementación de un generador de casos de prueba para procedimientos en lenguaje java se caracterizó por usar una metodología de desarrollo ágil, lo que facilitó su construcción.

4. Después de realizar el estudio concluimos que, la implementación de un generador de casos de prueba para procedimientos en lenguaje java aumenta significativamente la cantidad de errores encontrados con respecto a pruebas manuales, por lo que mejora la calidad del software desarrollado.

## Recomendaciones

1. A las personas relacionadas al ámbito de pruebas de software, les recomiendo no olvidar que los errores de software sólo son visibles una vez son encontrados, por lo que realizar pruebas desde la fase de codificación, es decir, pruebas unitarias, es una gran ventaja que mitiga posibles problemas futuros con el funcionamiento de los sistemas desarrollados.

2. Se recomienda a futuros investigadores en el tema la complementación de este trabajo incluyendo otras técnicas de generación de casos de prueba y la ampliación del ámbito a otros tipos de pruebas como lo son las pruebas de integración o punto a punto.

3. Se recomienda a los programadores y testers mantener un código limpio para que se puedan realizar pruebas unitarias, ya que no siempre estas se pueden realizar.

4. Se recomienda a los equipos de desarrollo usar prácticas ágiles para desarrollar software ya que mejoran el proceso de entrega de valor al usuario final, manteniendo a la vista siempre prioridades y objetivos.

5. Por último, se recomienda a los equipos de desarrollo complementar el desarrollo de software con conocimientos técnicos, como el desarrollo guiado por pruebas o el desarrollo guiado por comportamiento, y herramientas automatizadas de mejora de código, checkstyle, entre otros; como lo son SonarQube y Selenium.

## Referencias bibliográficas


- Andreu, R. (1996). *Estrategia Y Sistemas De Información.* Madrid: Mcgraw-hill.
- Barrientos, L. P. (2014). *Enfoque para Pruebas de Unidad Basado en la Generación Aleatoria de Objetos.* La Plata: Facultad de Informática - Universidad Nacional de La Plata.
- Bautista, G. H. (21 de Marzo de 2013). Obtenido de Metodologias de desarrollo: http://ithgermanhdez.blogspot.pe/2013/03/metodologias-de-desarrollo.html
- Blanco, R., Díaz, E., & Tuya, J. (2006). Generación automática de casos de prueba mediante busqueda dispersa. *Revista Española de Innovación, Calidad e Ingeniería del Software*, 24-35.
- EcuRed. (s.f.). Obtenido de EcuRed - Conocimiento con todos y para todos.
- EcuRed. (12 de Abril de 2012). *Prueba de Caja Blanca.* Obtenido de EcuRed - Conocimiento con todos y para todos: http://www.ecured.cu/Pruebas_de_caja_blanca
- EcuRed. (29 de Mayo de 2013). *Pruebas de software.* Obtenido de EcuRed - Conocimiento con todos y para todos: http://www.ecured.cu/Pruebas_de_software
- EcuRed. (s.f.). *Software*. Obtenido de EcuRed - Conocimiento con todos y para todos: http://www.ecured.cu/Software
- Esmite, I., Farías, M., & Farías, N. (2007). *Automatización y Gestión de las Pruebas Funcionales usando Herramientas Open Source.* Montevideo.
- *FunkLoad*. (s.f.). Obtenido de http://funkload.nuxeo.org/

- Gutiérrez, J. J., Escalona, M. J., Mejías, M., Torres, J., & Torres-Zenteno, A. (2007). *Generación automática de objetivos de prueba a partir de casos de uso mediante partición de categorías y variables operacionales.* Sevilla.
- Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M. (2010). *Metodología de la investigación.* México D.F.: McGRAW-HILL.
- IEEE Std. (1993). *IEEE Software Engineering Standard: Glossary of Software Engineering Terminology.* IEEE Computer Society Press.
- J. Pino, F. G. (2003). *Adaptación de las normas ISO/IEC 12207:2002 e ISO/IEC 15504:2003 para la evaluación de la madurez de procesos software en países en desarrollo.*
- Lamancha, B. P. (2009). Generación automática de casos de prueba para Líneas de Producto de Software. *Revista Española de Innovación, Calidad e Ingeniería del Software*, 17-27.
- Navarro, N., & Garzás, J. (4 de Junio de 2015). *¿Por qué utilizamos Puntos Historia para estimar y no horas?* Obtenido de javiergarzas.com: http://www.javiergarzas.com/2015/06/puntos-historia-para-estimar-y-no-horas.html
- Paez, N. (Julio de 2015). *GitBook.* Obtenido de Automatizacion de pruebas: https://nicopaez.gitbooks.io/test-automation/content/cobertura.html
- Pressman, R. S. (2010). *Ingeniería del software. Un enfoque práctico.* México D.F.: McGraw-Hill.
- Pumarejo, J. (2002). *La Industria de Software en el Perú, Análisis de Mercado y Estratégico Sectorial.*

- Quito Rodriguez, C. Z., & León García, T. (2013). *Guía de investigación.* Piura.
- SOFTENG. (s.f.). *Metodología Scrum para desarrollo de software - aplicaciones complejas.* Obtenido de Metodología Scrum: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html
- SOFTENG. (s.f.). *Proceso y Roles de Scrum.* Obtenido de The internet development company: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html
- Toledo, F. (6 de Junio de 2011). *Generación automática de datos de prueba.* Obtenido de Abstracta Blog - Simplificando el Testing: http://blog.abstracta.com.uy/2011/06/generacion-automatica-de-datos-de.html
- Trasobares, A. H. (s.f.). *Los sistemas de información: Evolución y Desarrollo.* Zaragoza: Departamento de Economía y Dirección de Empresas - Universidad de Zaragoza.
- Valdéz, C. J. (2014). *Implementación del modelo integral colaborativo (MDSIC) como fuente de innovación para el desarrollo ágil de software en las empresas de la zona centro - occidente en México.* Puebla.
- Vergara González, D. B. (2008). *Mejora del proceso software de una pequña empresa desarrolladora de software: Caso Competisoft-Perú Lambda*. Lima.

## Anexos

### Anexo 1: Encuesta De Investigación

ENCUESTA PARA MEDIR EL MODO EN QUE SE TRABAJAN CON LAS PRUEBAS DE SOFTWARE EN LAS EMPRESAS DE DESARROLLO

1. Actualmente en la empresa en la que labora, ¿Cuantos sistemas están en producción?
    a. 1
    b. Entre 1 y 5
    c. Entre 5 y 10
    d. Más de 10
2. En la empresa, ¿existe personal dedicado exclusivamente a realizar pruebas?
    a. Sí
    b. No
3. ¿Cuál es el tiempo a la semana que se dedica a realizar pruebas?
    a. Menos de 10 horas
    b. Entre 10 y 20 horas
    c. Entre 20 y 30 horas
    d. Más de 30 horas
4. Antes de lanzar una nueva versión de un sistema, ¿Qué tipos de pruebas se realizan?
    ☐ Pruebas unitarias
    ☐ Pruebas de componentes
    ☐ Pruebas de integración
    ☐ Pruebas de sistema
    ☐ Otros:

    …….

5. Durante el proceso de pruebas, que resultados son guardados

    a. No se guardan datos
    b. Solo errores
    c. Información detallada
    d. Otros:

6. Sobre el reporte de errores de los sistemas en producción

    a. No se guardan datos
    b. Solo errores críticos
    c. Información detallada
    d. Otros:

       …….

7. ¿Cuál es en promedio la cantidad de errores que se reportan por semana?

    a. Menos de 3
    b. Entre 3 y 20
    c. Entre 20 y 50
    d. Más de 50

8. ¿Cuál suele ser el tipo de error más común que se reporta?

    a. Errores de datos
    b. Errores de programación
    c. Errores de red
    d. Errores de hardware
    e. Otros:

9. ¿Cuál es la política de ejecución de pruebas?

    a. Manual
    b. Semiautomática
    c. Completamente automatizada
    d. Otros:

10. ¿Tiene conocimiento sobre alguna de las siguientes herramientas?

- ☐ Serenity BDD
- ☐ SonarQube
- ☐ JUNIT
- ☐ Selenium

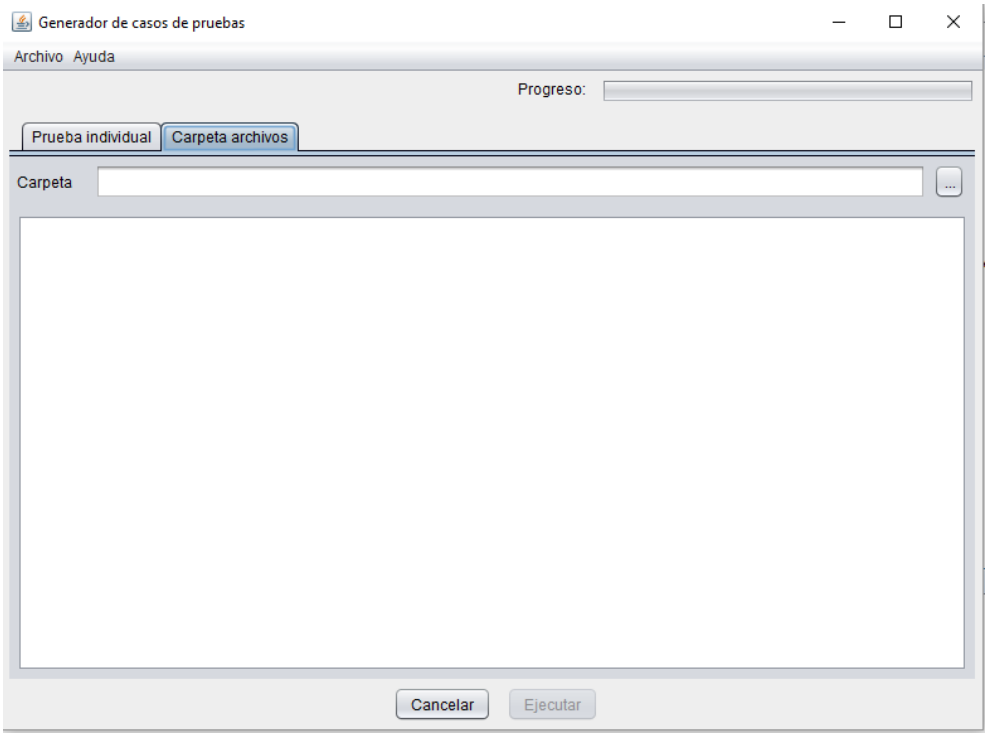
11. ¿Tiene conocimiento sobre el desarrollo guiado por pruebas o TDD?

a. No
b. Si, solo teórico
c. Sí, he trabajado con ello

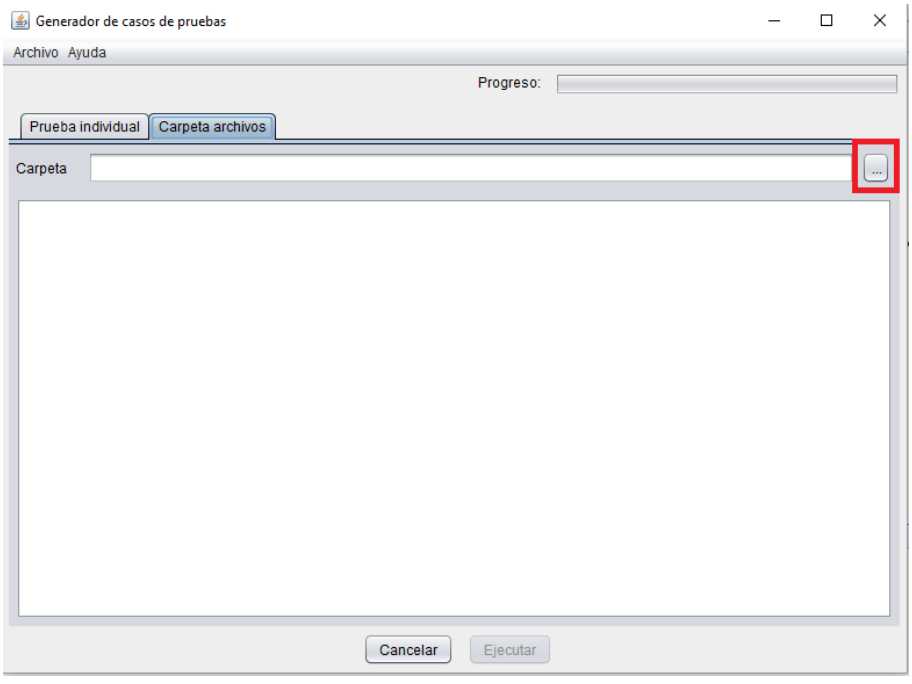
## Anexo 2: Flujo de ejecución del Generador de Casos de Pruebas

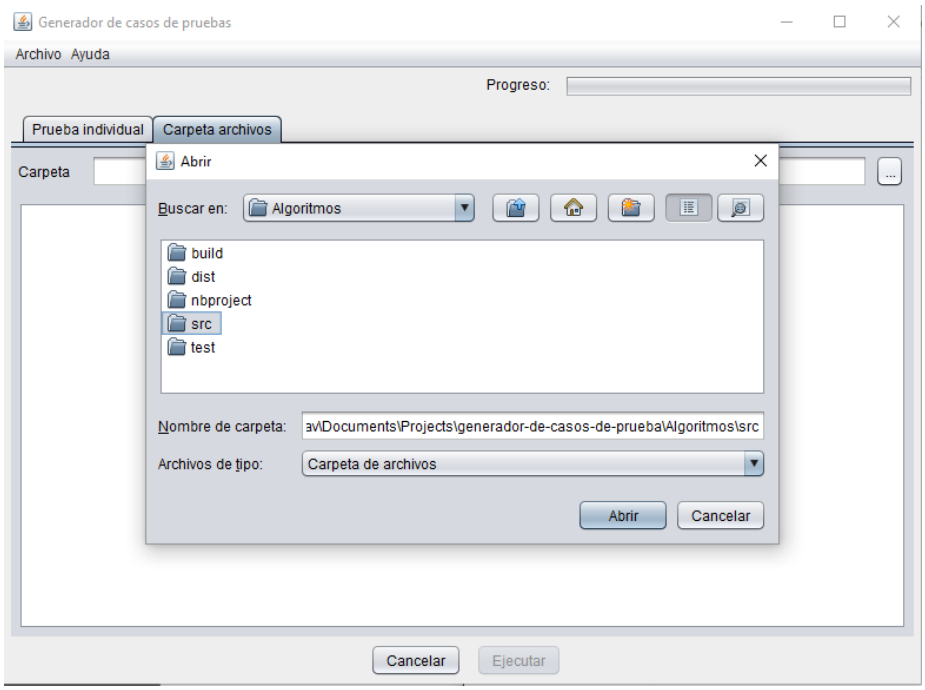
A continuación, se muestra el flujo de generación de casos de pruebas con pantallas para el código que fue usado en este trabajo de investigación.

1. Pantalla de inicio del generador.

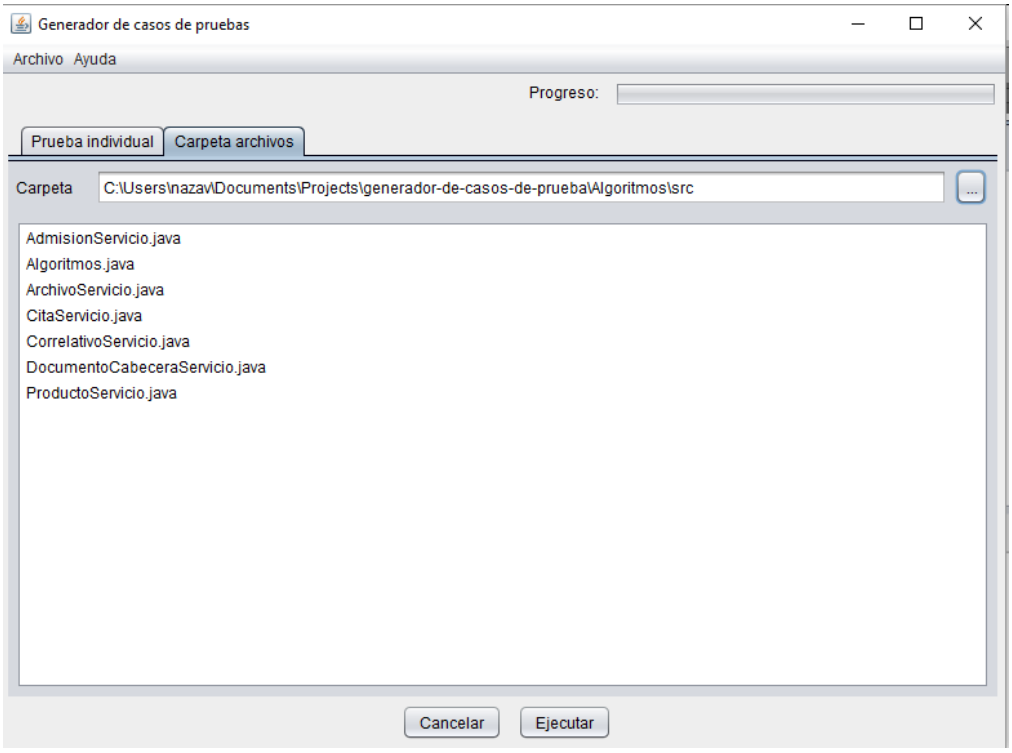



2. Dado que ejecutaremos un conjunto de archivos estructurados en forma de paquetes, el primer paso es abrir la carpeta que contenga el código fuente, tal como se muestra en las siguientes imágenes:

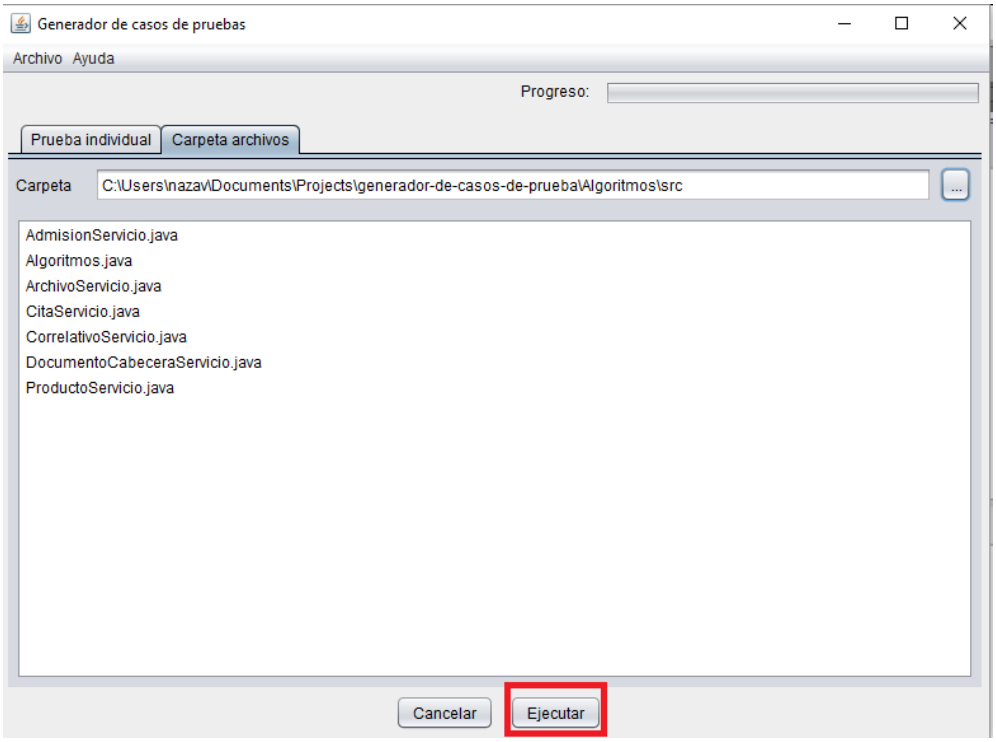




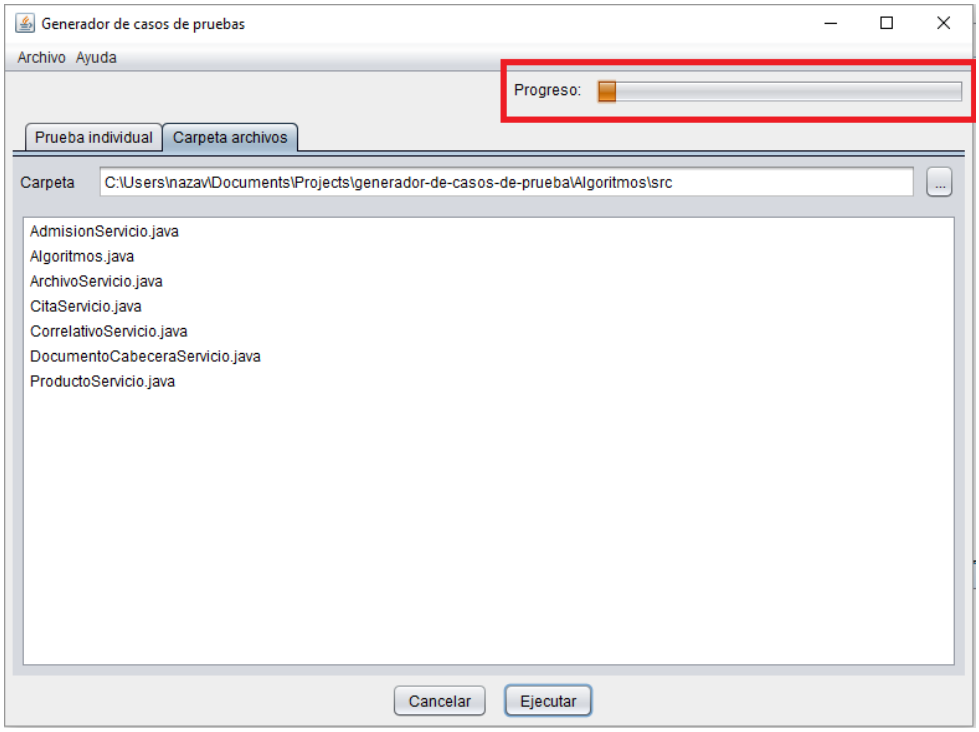

3. Una vez seleccionada la carpeta, la pantalla mostrará una lista con los archivos que tienen por los menos un método sobre el cual se pueden generar casos de prueba.



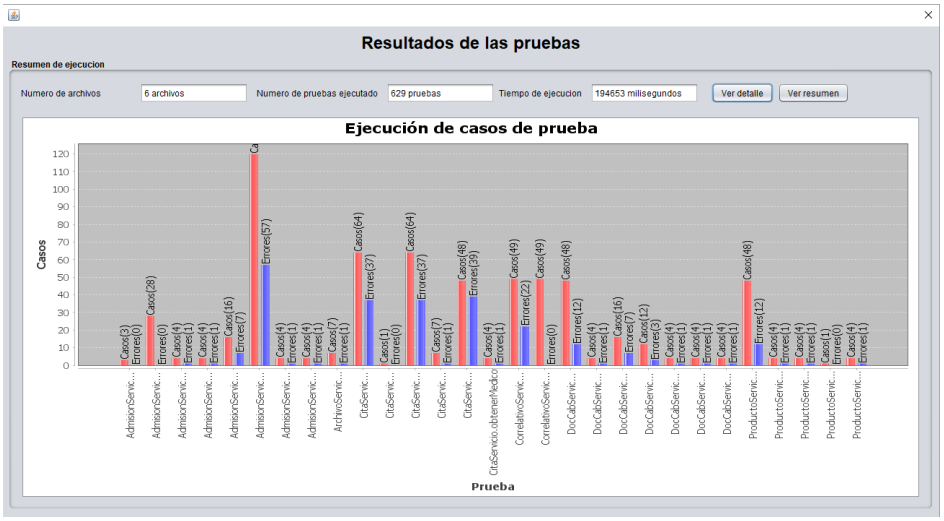
4. El siguiente paso es ejecutar el generador, lo que iniciará el proceso que se ha descrito en los capítulos anteriores.



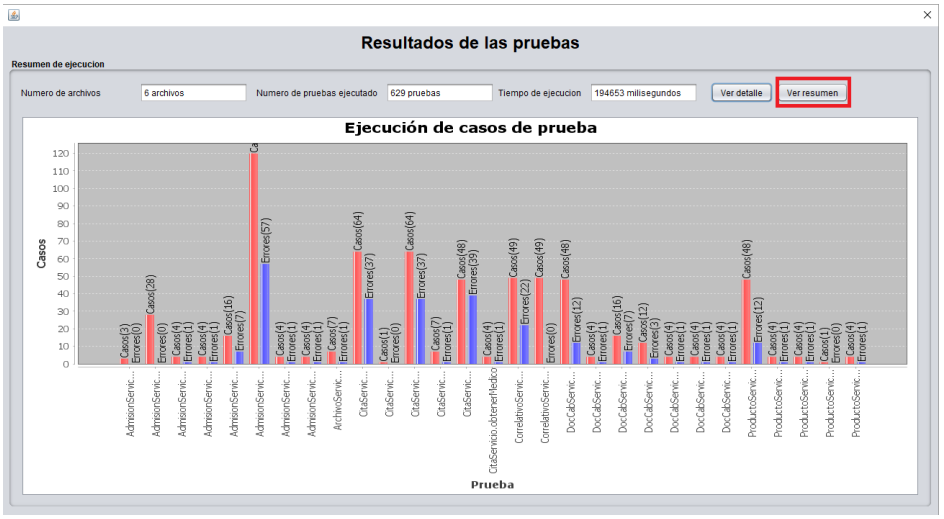

5. Conforme el proceso va avanzando, se muestra el progreso en la barra ubicada en la parte superior derecha



6. Una vez finalizada la ejecución, se mostrará la siguiente pantalla con un resumen de los resultados de manera automática.

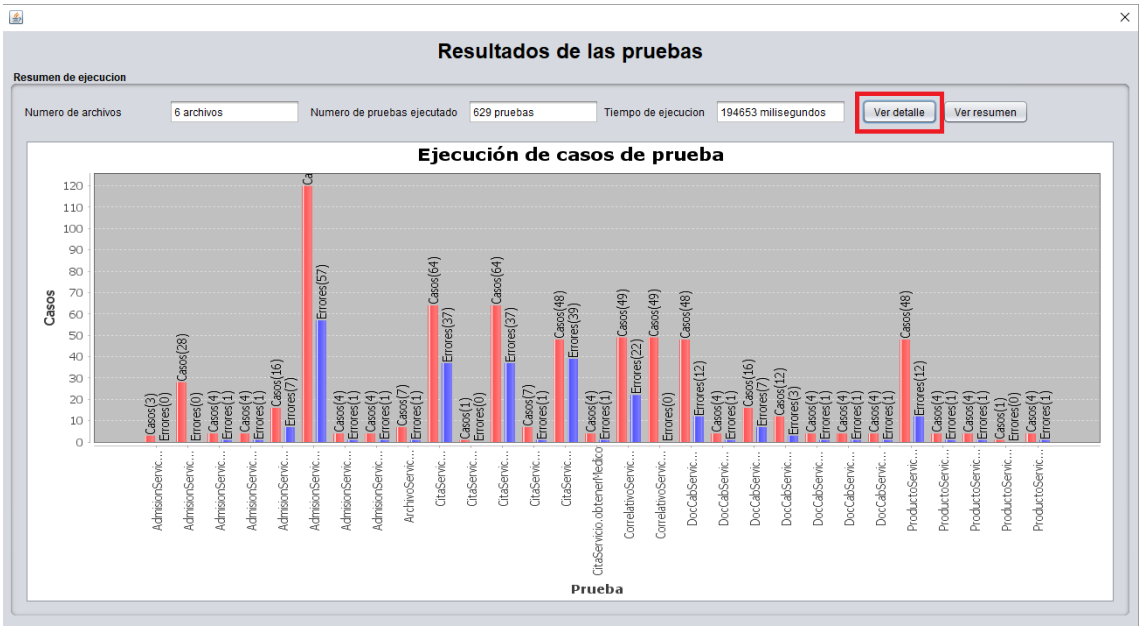



7. Desde esta pantalla se pueden ver los resultados resumidos en éxito/error presionando el botón resaltado.

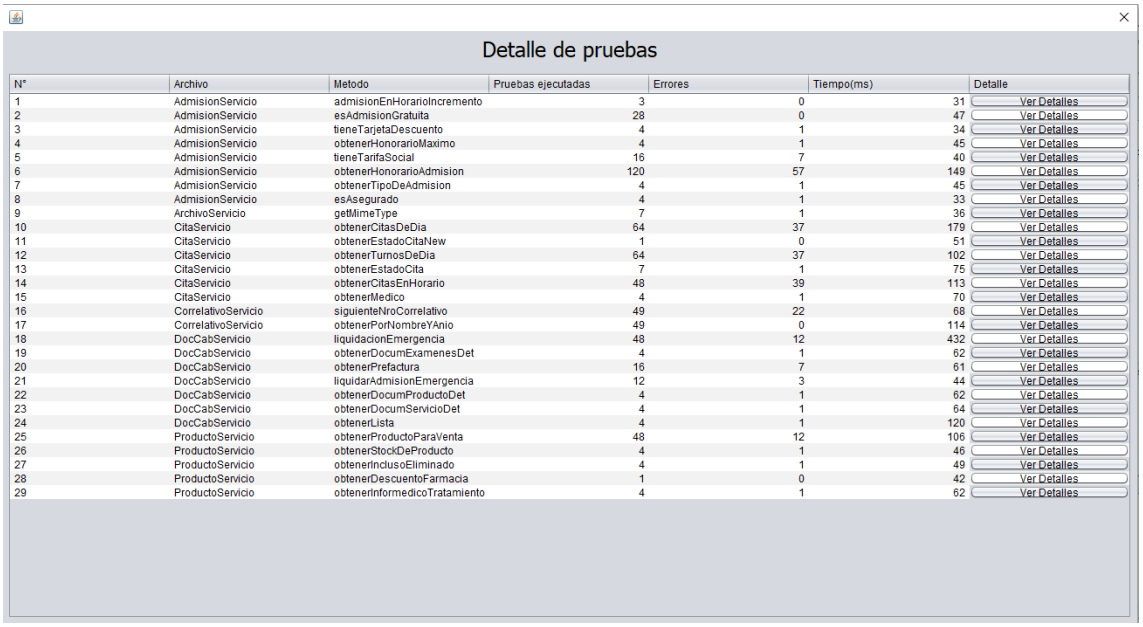



8. Con lo que se muestra un gráfico de torta con los casos de éxito/error.

9. O bien, se puede mostrar el detalle de la ejecución con el botón que está al lado del anterior, resaltado en la siguiente imagen:



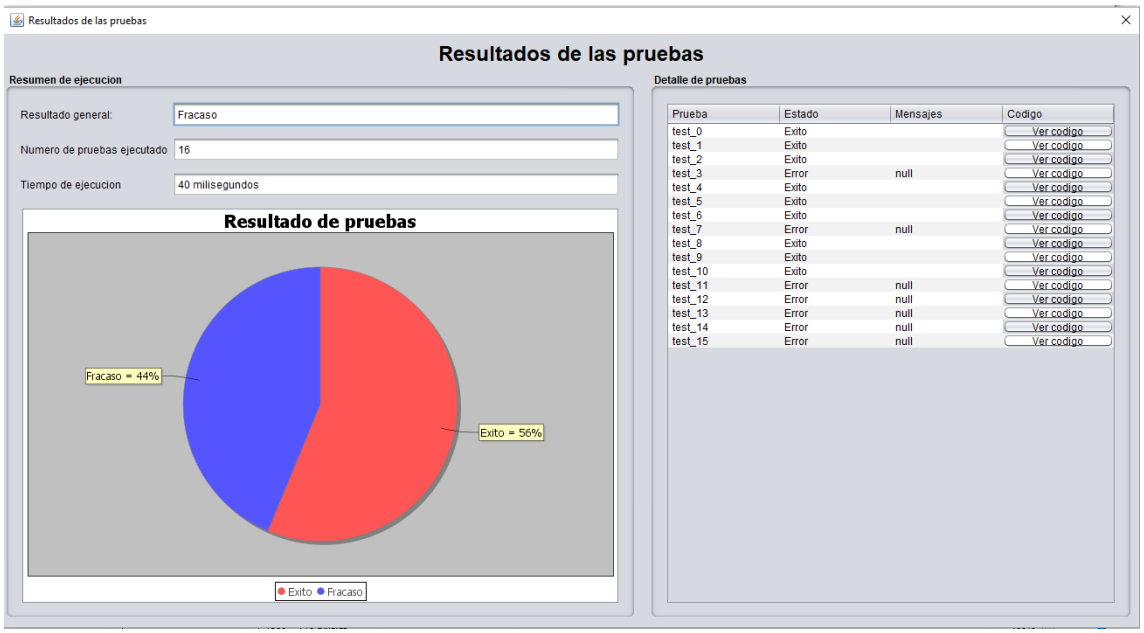

10. Lo que nos lleva a la siguiente pantalla, la cuál es la pantalla más importante para esta investigación, ya que resume los resultados de los casos de pruebas, errores y tiempo que toma ejecutarlos.



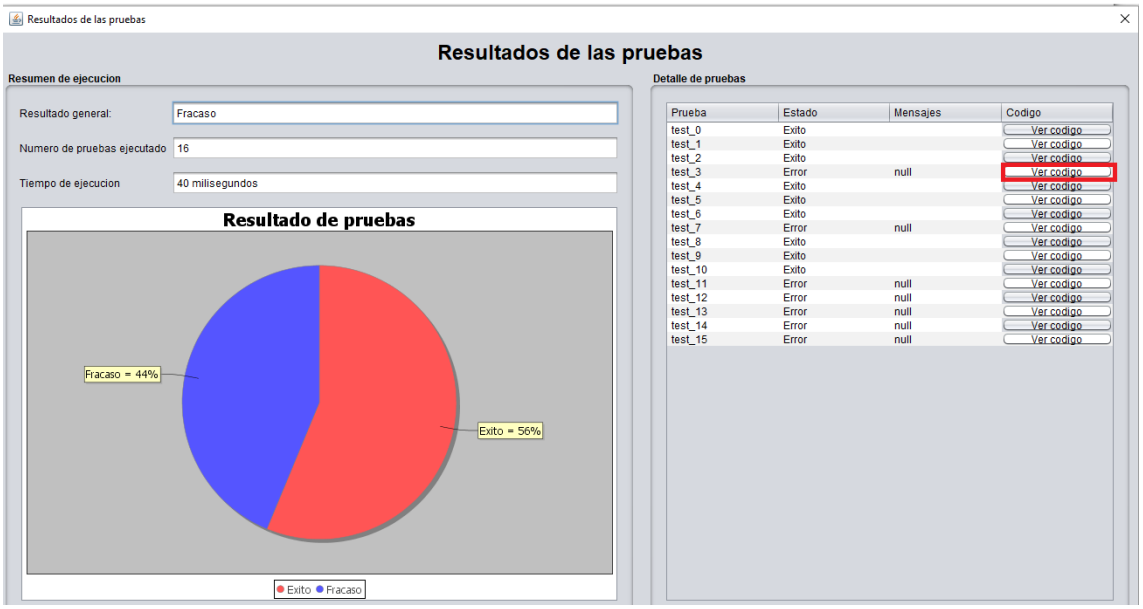

| N° | Archivo | Metodo | Pruebas ejecutadas | Errores | Tiempo(ms) | Detalle |
|---|---|---|---|---|---|---|
| 1 | AdmisionServicio | admisionEnHorarioIncremento | 3 | 0 | 31 | Ver Detalles |
| 2 | AdmisionServicio | esAdmisionGratuita | 28 | 0 | 47 | Ver Detalles |
| 3 | AdmisionServicio | tieneTarjetaDescuento | 4 | 1 | 34 | Ver Detalles |
| 4 | AdmisionServicio | obtenerHonorarioMaximo | 4 | 1 | 45 | Ver Detalles |
| 5 | AdmisionServicio | tieneTarifaSocial | 16 | 7 | 40 | Ver Detalles |
| 6 | AdmisionServicio | obtenerHonorarioAdmision | 120 | 57 | 149 | Ver Detalles |
| 7 | AdmisionServicio | obtenerTipoDeAdmision | 4 | 1 | 45 | Ver Detalles |
| 8 | AdmisionServicio | esAsegurado | 4 | 1 | 33 | Ver Detalles |
| 9 | ArchivoServicio | getMimeType | 7 | 1 | 36 | Ver Detalles |
| 10 | CitaServicio | obtenerCitasDeDia | 64 | 37 | 179 | Ver Detalles |
| 11 | CitaServicio | obtenerEstadoCitaNew | 1 | 0 | 51 | Ver Detalles |
| 12 | CitaServicio | obtenerTurnosDeDia | 64 | 37 | 102 | Ver Detalles |
| 13 | CitaServicio | obtenerEstadoCita | 7 | 1 | 75 | Ver Detalles |
| 14 | CitaServicio | obtenerCitasEnHorario | 48 | 39 | 113 | Ver Detalles |
| 15 | CitaServicio | obtenerMedico | 4 | 1 | 70 | Ver Detalles |
| 16 | CorrelativoServicio | siguienteNroCorrelativo | 49 | 22 | 68 | Ver Detalles |
| 17 | CorrelativoServicio | obtenerPorNombreYAnio | 49 | 0 | 114 | Ver Detalles |
| 18 | DocCabServicio | liquidacionEmergencia | 48 | 12 | 432 | Ver Detalles |
| 19 | DocCabServicio | obtenerDocumExamenesDet | 4 | 1 | 62 | Ver Detalles |
| 20 | DocCabServicio | obtenerPrefactura | 16 | 7 | 61 | Ver Detalles |
| 21 | DocCabServicio | liquidarAdmisionEmergencia | 12 | 3 | 44 | Ver Detalles |
| 22 | DocCabServicio | obtenerDocumProductoDet | 4 | 1 | 62 | Ver Detalles |
| 23 | DocCabServicio | obtenerDocumServicioDet | 4 | 1 | 64 | Ver Detalles |
| 24 | DocCabServicio | obtenerLista | 4 | 1 | 120 | Ver Detalles |
| 25 | ProductoServicio | obtenerProductoParaVenta | 48 | 12 | 106 | Ver Detalles |
| 26 | ProductoServicio | obtenerStockDeProducto | 4 | 1 | 46 | Ver Detalles |
| 27 | ProductoServicio | obtenerInclusoEliminado | 4 | 1 | 49 | Ver Detalles |
| 28 | ProductoServicio | obtenerDescuentoFarmacia | 1 | 0 | 42 | Ver Detalles |
| 29 | ProductoServicio | obtenerInformedicoTratamiento | 4 | 1 | 62 | Ver Detalles |

11. Para ver más detalle de cada caso generado y/o ejecutado, se puede pulsar el botón “Ver detalle”, resaltado en la imagen.


12. Esta acción nos mostrará los resultados de la ejecución para ese método en específico, a la izquierda el resumen en un gráfico de torta con la proporción de errores y valores de resumen (resultado general, número de pruebas y tiempo de ejecución), y; en la parte derecha el detalle de cada caso de prueba creado automáticamente.



13. Se puede ver el código fuente de cada caso de prueba creado para más detalle de su ejecución, de tal manera que este pueda ser replicado y el método depurado para su correcta implementación.




14. A continuación, se muestra el código fuente Java para una prueba unitaria.