

**UNIVERSIDAD AUTÓNOMA “GABRIEL RENE MORENO”  
FACULTAD DE INGENIERÍA EN CIENCIAS DE LA  
COMPUTACIÓN Y TELECOMUNICACIONES**

**“UAGRM SCHOOL OF ENGINEERING”**



**MAESTRÍA EN INGENIERÍA DE *SOFTWARE***

**“MEJORA EN LA CURVA DE TIEMPO DE EJECUCIÓN  
DE PROYECTOS DE SOFTWARE CON LA  
INTEGRACIÓN DE HERRAMIENTAS PARA LA  
GENERACIÓN AUTOMÁTICA DE CASOS DE PRUEBA,  
EN DUALBIZ S.R.L.”**

TRABAJO FINAL DE GRADO BAJO LA MODALIDAD DE TESIS PARA OPTAR AL  
TÍTULO DE MAESTRO EN CIENCIAS

**AUTOR:**

Ing. Alcides Yohacin Leaños Rodriguez

**DIRECTOR DE TRABAJO FINAL DE GRADO:**

Alida Nersa Paneque Ginarte Ph.D

Santa Cruz, Bolivia  
Julio, 2018

## **Cesión de derechos**

Declaro bajo juramento que el trabajo aquí descrito, titulado “**Mejora en la curva de tiempo de ejecución de proyectos de *software* con la integración de herramientas para la Generación Automática de Casos de Prueba en Dualbiz S.R.L.**” es de propia autoría; que no ha sido previamente presentada para ningún grado de calificación profesional; y, que se ha consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaro que cedo mi derecho de propiedad Intelectual correspondiente a este trabajo, a la UAGRM Facultad de Ingeniería en Ciencias de la Computación y Telecomunicaciones, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

---

**Ing. Alcides Yohacin Leños Rodríguez**

## **Dedicatoria**

A mis padres, que con su ejemplo me mostraron que siempre hay camino por recorrer; algo que aprender y algo que enseñar.

A mi hermana y mis sobrinos por su apoyo y cariño que siempre irradian, son un motor para continuar siempre.

## **Agradecimiento**

A Dios el compás que guía mis pasos, el horizonte y la fuente de fortaleza que me permite culminar un objetivo más en la vida.

A Alida Paneque Ginarte Ph.D. Corazón noble y desinteresado, que ayudó mucho con su tiempo y conocimientos para realizar la presente investigación.

A José Miguel Rojas Ph.D. Gracias a su investigación sobre herramientas de automatización en la Universidad Sheffield, Inglaterra, pude encaminar el desarrollo de la presente investigación.

Al personal de Dualbiz S.R.L, por ayudarme en la ejecución de la investigación, a todos los chicos que aportaron con su esfuerzo para la culminación del proyecto.

A la Escuela de Ingeniería de la UAGRM, por su labor de compartir el conocimiento y las herramientas necesarias para dar continuidad a nuestra formación profesional.

A la Ing. Daniela Osinaga, quien mencionó: “Uno solo camina más rápido pero acompañado se llega más lejos”, gracias por ayudarme a llegar más lejos.

## RESUMEN

La presente investigación responde al problema científico ¿Cómo mejorar la curva de tiempo a través de la detección temprana de errores en la fase de codificación con Herramientas para la Generación Automática de Casos de Prueba en los proyectos de desarrollo de *software* para Dualbiz S.R.L? Reconoce como objetivo general: Evaluar la integración de una herramienta para la Generación automática de Casos de Pruebas para el contexto mencionado.

El marco teórico se construye a partir de la sistematización realizada a diferentes autores que sustentan, teóricamente, las Herramientas para la Generación Automática de Casos de Prueba; se define *Intellitest* como la herramienta a utilizar en la propuesta de solución. Los resultados de la aplicación de los instrumentos de investigación (entrevista y encuesta) se procesan utilizando la Matriz de Vester y se determina la correlación de los problemas críticos, “Los tiempo de entrega no siempre se respetan” y “Las pruebas internas consumen un tiempo considerable en la codificación”.

A partir del diagnóstico, se estructura la propuesta de integración de la herramienta seleccionada *IntelliTest* dentro de la fase de Codificación de Proyectos de *Software* en Dualbiz S.R.L., se afecta primero los proyectos desarrollados en .NET por ser éstos los que están en actual fase de codificación y tener un conjunto de requerimientos pendientes de ejecución. Finalmente, se valida los resultados de la integración mediante la Prueba t de *Student* para muestras relacionadas, comprobándose la mejora en la curva de tiempo de ejecución de proyectos de desarrollo de *software* en fase de codificación para el contexto que se investiga.

## ABSTRACT

This research responds to the scientific problem, How to improve the time curve through the early detection of errors in coding phase with Tools for the Automatic Test Cases Generation in the *software* development stage for Dualbiz S.R.L? Recognizes as general objective: To evaluate the integration of a tool for the Automatic Test Cases Generation for the mentioned context.

The theoretical framework is constructed from the systematization made to different authors who theoretically support the Tools for the Automatic Generation of Test Cases; Intellitest is defined as the tool to be used in the solution proposal. The results of the application of the research instruments (interview and survey) are processed using the Vester Matrix and the correlation of the critical problems is determined, "Delivery times are not always respected" and "Internal tests consume a considerable time in the coding".

Based on the diagnosis, the integration proposal of the selected tool IntelliTest is structured within the Coding phase of *Software* Projects in Dualbiz SRL, the projects developed in .NET are affected first, as these are in the current coding phase and have a set of pending enforcement requirements. Finally, the results of the integration are validated by Student's t-test, checking the improvement in the execution time curve of *software* development projects in the coding phase for the context under investigation.

## ÍNDICE GENERAL

INTRODUCCIÓN .....	1
1. Antecedentes del Problema .....	4
2. Planteamiento del problema .....	6
2.1 Objeto de Estudio .....	6
2.2 Campo de Acción .....	7
3. Objetivos .....	7
3.1 Objetivo General.....	7
3.2 Objetivos específicos.....	7
4. Idea Científica a Defender .....	8
5. Justificación .....	8
6. Delimitación de la investigación.....	9
7. Diseño Metodológico .....	9
7.1. Tipo de Investigación.....	9
7.2. Métodos de Investigación.....	10
7.3. Técnicas e Instrumentos de Investigación.....	10
7.4. Población y Muestra .....	11
1. CAPÍTULO I. MARCO TEÓRICO Y CONCEPTUAL .....	12
1.1. Acercamiento al concepto de pruebas .....	12
1.2. Pruebas de <i>Software</i> .....	16
1.3. El espectro de las pruebas .....	20
1.4. Prueba Unitaria.....	21
1.5. Generación Automática de Casos de Prueba.....	25
1.5.1. Análisis de caminos posibles.....	29
1.5.2. Ingeniería basada en búsqueda .....	31
1.5.3. Cálculo de la distancia asociada a una solución .....	34
1.6. <i>IntelliTest</i> .....	36
2. CAPITULO II - DIAGNÓSTICO.....	41
2.1. Acercamiento al contexto que se investiga .....	41
2.1.1. Estructura organizacional de la empresa .....	41
2.1.2. Equipo de Desarrollo .....	42

2.1.3. Fase de codificación y pruebas .....	43
2.2. Procedimiento para el Diagnóstico .....	45
2.2.1. Diseño de la Encuesta y Entrevista .....	48
2.2.2. Elaboración de Matriz de Vester.....	51
2.3 Conclusiones.....	61
3. CAPÍTULO III – PROPUESTA.....	62
3.1. Propuesta .....	62
3.1.1. Por qué <i>Intellitest</i> como herramienta de integración.....	62
3.1.2. Pasos para la integración .....	63
3.1.3. Plan de contingencia .....	65
3.2. Procedimiento para el Experimento.....	66
3.3. Proceso Experimental .....	69
3.3.1. Definición del alcance.....	69
3.3.2. Planificación .....	69
3.3.3. Operación.....	77
3.2.4. Análisis e Interpretación .....	82
3 CONCLUSIONES .....	84
4 RECOMENDACIONES.....	85
5 REFERENCIA BIBLIOGRÁFICA .....	86
6 BIBLIOGRAFÍA.....	89
7 ANEXOS.....	90



## ÍNDICE DE CUADROS

<b>Cuadro No.1.</b> Entrevista a Gerencia de Proyectos Dualbiz S.R.L. ....	48
<b>Cuadro No.2.</b> Diseño de la Encuesta a Equipo de Desarrollo Dualbiz S.R.L. ....	49
<b>Cuadro No.3</b> Diseño Matriz de Vester, caso Dualbiz S.R.L.....	51
<b>Cuadro No.4.</b> Matriz refinada de problemas para elaboracion de Matriz de Vester .	52
<b>Cuadro No.5.</b> Ficha técnica P1.....	53
<b>Cuadro No.6.</b> Ficha Técnica P2 .....	53
<b>Cuadro No.7.</b> Ficha Técnica P3 .....	53
<b>Cuadro No. 8.</b> Ficha Técnica P4 .....	54
<b>Cuadro No.9.</b> Ficha Técnica P5 .....	54
<b>Cuadro No.10.</b> Ficha Técnica P6 .....	54
<b>Cuadro No.11.</b> Ficha Técnica P7 .....	54
<b>Cuadro No.12.</b> Ficha Técnica P8 .....	55
<b>Cuadro No.13.</b> Ficha Técnica P9 .....	55
<b>Cuadro No.14.</b> Ficha Técnica P10 .....	55
<b>Cuadro No.15.</b> Relación de Influencia para Matriz Vester.....	56
<b>Cuadro No.16.</b> Matriz de Dependencia e Influencia .....	57
<b>Cuadro No.17.</b> Tabla de coordenadas y promedio dependencia e Influencia .....	59
<b>Cuadro No. 18.</b> Cuadrante resultado de Problemas, en Dualbiz S.R.L.....	60
<b>Cuadro No.19.</b> Hoja de trabajo <i>IntelliTest</i> .....	65
<b>Cuadro No.20.</b> Requerimientos iniciales del proyecto .....	71
<b>Cuadro No.21.</b> Criterio de complejidad para balancear requerimientos .....	72
<b>Cuadro No.22.</b> Requerimientos balanceados contexto O <sub>1</sub> .....	74
<b>Cuadro No.23.</b> Requerimientos balanceados contexto O <sub>2</sub> .....	75
<b>Cuadro No.24.</b> Distribución Normal contexto O <sub>1</sub> .....	79
<b>Cuadro No. 25.</b> Distribución Normal contexto O <sub>2</sub> .....	80
<b>Cuadro No.26.</b> Resultados <i>Excel</i> . Prueba t de <i>Student</i> para muestras emparejadas .....	81

## ÍNDICE FIGURAS

<b>Figura No.1.</b> Tipos de prueba vs peso ponderado en tiempo de desarrollo .....	6
<b>Figura No.2.</b> Espiral de pruebas en el ciclo de vida del <i>software</i> .....	18
<b>Figura No.3.</b> Un Modelo de Proceso de Pruebas .....	19
<b>Figura No.4.</b> Ilustración de una prueba básica .....	22
<b>Figura No.5.</b> Clase a probar Escrita en C#.....	23
<b>Figura No.6.</b> Prueba unitaria Realizada en .NET c# con <i>MSTest FrameWork</i> .....	24
<b>Figura No.7.</b> Prueba unitaria escrita con <i>Junit Framework</i> .....	24
<b>Figura No.8.</b> Ejemplo de análisis de cobertura clase <i>&lt;Stack&gt;</i> .....	26
<b>Figura No.9.</b> Suite de Pruebas utilizado para el análisis de la cobertura de la clase pila.....	27
<b>Figura No.10.</b> FPA Básico de la Ejecución Simbólica .....	30
<b>Figura No.11.</b> Información almacenada por TCSS en Grafo de control .....	33
<b>Figura No.12.</b> <i>TraceGen</i> algoritmo Generador de Soluciones.....	35
<b>Figura No.13.</b> Ejecutar <i>Intellitest</i> (Generación Automática de Pruebas) .....	38
<b>Figura No.14.</b> Resultados de Exploración <i>IntelliTest</i> .....	39
<b>Figura No.15.</b> Ejecución <i>IntelliTest</i> para una Clase completa .....	39
<b>Figura No.16.</b> Menú emergente Creación de Proyecto de Pruebas <i>IntelliTest</i> .....	40
<b>Figura No.17.</b> Estructura Organizacional Dualbiz S.R.L.....	42
<b>Figura No.18.</b> Equipos de Desarrollo por Tecnología, Dualbiz S.R.L. ....	43
<b>Figura No.19.</b> Flujo de trabajo <i>Scrum</i> .....	44
<b>Figura No.20.</b> Cuadrante resultante de problemas Matriz de Vester.....	47
<b>Figura No.21.</b> (a) Conocimiento de Casos de Prueba. (b) Nivel de Conocimiento acerca de Casos de Prueba .....	50
<b>Figura No.22.</b> Ponderación de los problemas relevados en la Encuesta y Entrevista .....	51
<b>Figura No.23.</b> Cantidad de valores asignados según ponderación .....	58
<b>Figura No.24.</b> Matriz de Vester del Caso Dualbiz S.R.L.....	60
<b>Figura No.25.</b> Etapas de Integración de Herramienta Propuesta <i>IntelliTest</i> .....	64

<b>Figura No.26.</b> Proceso Experimental.....	67
<b>Figura No.27.</b> Distribución Normal de la muestra contexto $O_1$ .....	79
<b>Figura No.28.</b> Distribución Normal de la muestra contexto $O_2$ .....	81
<b>Figura No.29.</b> Distribución Normal de la muestra contexto $O_2$ .....	83

## ÍNDICE DE ANEXOS

<b>Anexo No.1.</b> Entrevista a Gerentes de Desarrollo y Operaciones.....	90
<b>Anexo No.2.</b> Encuesta a desarrolladores .....	92
<b>Anexo No.3.</b> Operacionalización del Objeto de Estudio .....	94
<b>Anexo No.4.</b> Ejemplo de análisis de cobertura de código para la Clase Dimisión (BrhOrgDim) .....	95
<b>Anexo No.5.</b> Integración <i>IntelliTest</i> sobre el Proyecto de Barrido .....	95
<b>Anexo No.6.</b> Cambios para compilación directa desde la herramienta y acceso a Base de Datos.....	96

## INTRODUCCIÓN

La Ingeniería del *Software* es una disciplina cuya meta es el desarrollo costeable de sistemas de *software* (este es abstracto e intangible). No está restringido o gobernado por leyes físicas o procesos de manufactura; de alguna forma, esto simplifica la Ingeniería del *Software* ya que no existen limitaciones físicas del potencial del *software*. Sin embargo, esta falta de restricciones naturales también significa que el *software* puede llegar a ser extremadamente complejo y por lo tanto muy difícil de entender. La noción de Ingeniería del *Software* fue propuesta inicialmente en 1968 para discutir lo que en ese momento se llamó la “Crisis del *Software*”. Esta crisis del *software* fue el resultado de la introducción de nuevas computadoras con *hardware* basados en circuitos integrados. Su poder hizo que las aplicaciones hasta entonces irrealizables, fueran una propuesta factible. El *software* resultante fue órdenes de magnitud más grande y más complejo que sistemas previos. Se necesitaban nuevas técnicas y métodos para controlar la complejidad inherente a los sistemas grandes (Sommerville, 2012).

Estas técnicas han llegado a ser parte de la Ingeniería del *Software* y son ampliamente utilizadas, desde 1968 comienza una carrera para la mejora de los procesos dentro de la Ingeniería del *Software* en sí mismo, ahora se conoce que “no existe un enfoque ideal” a la Ingeniería del *Software*, existen una variedad dependiente de los diferentes sistemas y organizaciones que utilizan estos sistemas. Desde un punto de vista más práctico se puede determinar que se pueden usar distintos enfoques, distintas herramientas dependiendo del *software*, el tipo de proyecto, la organización y el uso final del mismo.

Ian Sommerville resume dos aspectos claves:

1. Disciplina de Ingeniería. Los ingenieros hacen que las cosas funcionen. Aplican teorías métodos y herramientas donde sea conveniente, pero las utilizan de forma selectiva y siempre tratando de descubrir soluciones a los problemas, aun cuando no existan teorías y métodos aplicables para resolverlos. Los ingenieros también saben que deben trabajar con restricciones financieras y organizacionales, por lo que buscan soluciones tomando en cuenta estas restricciones.
2. Todos los aspectos de la producción de software. La Ingeniería del *Software* no solo comprende los procesos técnicos del desarrollo de *software*, sino también otras actividades tales como la gestión del proyectos de *software* y el desarrollo de herramientas, métodos y teorías de apoyo a la producción de *software*.

Como todo proceso de construcción, el *software* comprende etapas que van desde el análisis donde se definen los lineamientos, aspectos organizacionales, funcionales, no funcionales (rasgos del sistema y el entorno en el cual se va a desenvolver), entre otros. Comprende toda una ingeniería de requerimientos, la programación o construcción que consiste en programar y probar los programas informáticos de *software* y la posterior instalación y evaluación del nuevo sistema de *software* no terminando el ciclo en la etapa anterior sino originando un ciclo de soporte y mantenimiento al mismo (Falgueras, 2003) . Todo este proceso no se encuentra libre de problemas y errores que pueden surgir en cualquier etapa del mismo.

El concepto de cero defectos y el programa de cero defectos son una plataforma o serie de conceptos desarrollada por el ingeniero en administración Philip B. Crosby. Bajo la consigna de “Evitar los errores y corregirlos desde sus inicios”, en lugar de buscar soluciones a defectos vistos posteriormente”. (Crosby, 2012) Tratando de explicar claramente la postura de Crosby se entiende que se debe evitar o corregir los errores lo más temprano posible en cualquiera de las etapas de los proyectos.

Dualbiz S.R.L. es una empresa de desarrollo de *software* establecida en la ciudad de Santa Cruz de la Sierra – Bolivia; con nueve años en el mercado y la visión de ser un proveedor de soluciones de negocio para empresas; comercializa *software* de gestión, aplicaciones móviles y otros proyectos relacionados en su mayoría al *software* de gestión “Dual ERP”.

La producción de *software* de calidad para sus clientes es su meta principal, nació como una empresa de *software* artesanal en sus primeros pasos; haciendo referencia a los problemas de la “Crisis del *Software*” citada por Ian Sommerville y se ha visto inmersa en un proceso constante de evolución tratando de cumplir con su meta principal. Proceso en el cual se ve inmerso en las complicaciones antes descritas, es decir: restricciones financieras, organizacionales, inclusión de herramientas que permitan una mejor gestión dentro de cada etapa de los proyectos, entre otros. Siempre con la consigna de la captura de la calidad bajo el concepto de “Conformidad de Requisitos y confiabilidad en el funcionamiento” (Lisa Crispin, 2009).

## 1. Antecedentes del Problema

Los primeros pasos de una empresa suelen ser un auténtico caos, enfrentándose a una multitud de tareas sin tener claro cuáles deben ser priorizadas, el orden correcto que deben desarrollarse o los tiempos de ejecución en el que deben ser finalizadas.

Dualbiz S.R.L. en el transcurso de vida como empresa, se ha visto implicada, constantemente, en una serie de cambios desde el ámbito organizacional, estructural y computacional. Iniciando con un equipo de 3 persona y un solo cliente, como parte de un grupo empresarial INGEMAQ, a la etapa actual, en la que atiende a clientes externos que suman un total de 30 con un equipo interdisciplinario de 13 personas y 6 proyectos bases que comercializa.

Con las restricciones financieras y organizacionales de cualquier emprendimiento de este ámbito, ha logrado desarrollar su propia metodología adaptando lineamientos de Metodologías Ágiles como XP, utilizando también buenas prácticas de *SCRUM* como marco de trabajo, introduciendo Herramientas de Gestión de Configuración, Gestión Documental y otros, siempre con la meta final de lograr cumplir los compromisos con sus clientes.

El mercado competitivo actual exige nuevos desafíos, tener entregables más ágiles, menor cantidad de errores y defectos, precio competitivos, entre otros aspectos. Es una meta, la mejora continua como empresa y para lograr esto es necesario optimizar los procesos internos (operativos y ejecutivos). Durante y después del proceso de implementación, el programa que se está desarrollando debe ser comprobado para asegurar que satisface su especificación y entrega la



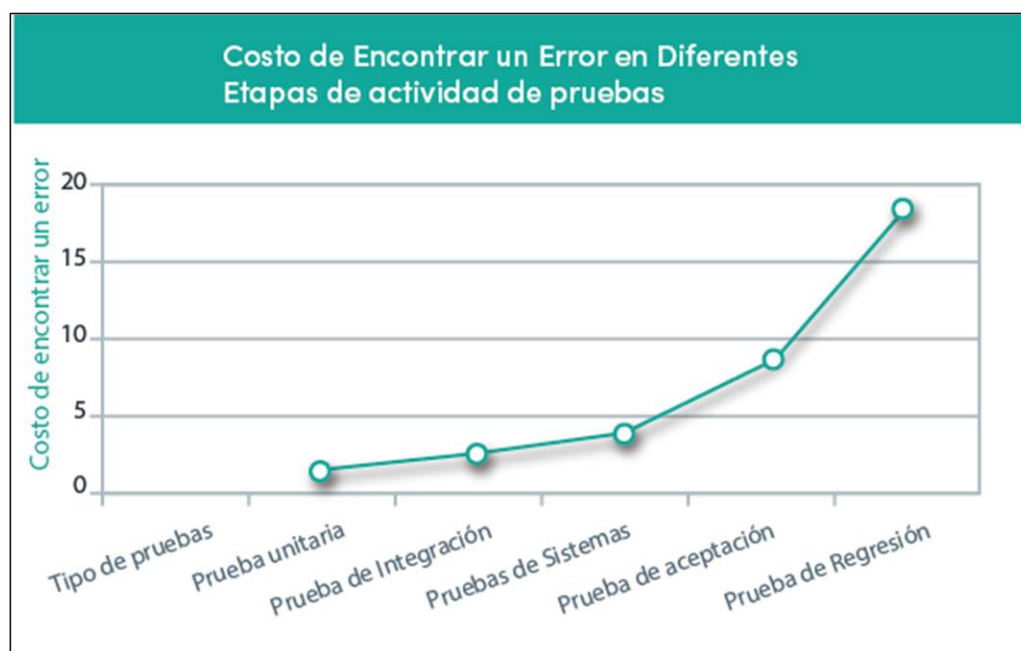
funcionalidad esperada. Generalmente, por lo tanto, el objetivo de las pruebas del *software* es convencer a los desarrolladores del sistema y a los clientes que el *software* es lo suficientemente bueno para su uso operacional.

La prueba es un proceso que proporciona confianza en el *software*. Si bien actualmente se encaran pruebas de entrega (realizadas por la persona que desarrolla) y pruebas de aceptación (realizadas con el cliente), existen productos finales entregados al cliente que no siempre cumplen las expectativas en tiempo y en calidad del mismo, recordando la cita anterior de Lisa Crispín “Calidad como conformidad en los requisitos y confiabilidad en el funcionamiento”.

Es decir, existen problemas en los entregables, los mismo pueden ser causados a partir de una mala gestión de los contratos (funcionalidades), sobrecarga de horas de trabajo por adquisición de nuevos compromisos causando presión en el equipo y retrasos en entregas, una mala gestión de los requerimientos, así como incidencias de mantenimiento por errores introducidos en la fase de codificación.

En la actualidad Dualbiz S.R.L. carece de casos de pruebas asociados al código y esta actividad no está incluida como hito dentro de la gestión de proyecto. Si bien el porcentaje de recubrimiento de código no asegura la calidad del producto, una de las definiciones de calidad es la de Crosby que se cita anteriormente, que asocia la calidad con el término “Cero Defectos” y con la idea de corregir errores desde los inicios. Se debe tomar en cuenta que actualmente los factores de prueba dependen en una primera fase por el programador, dependiendo del conocimiento de la sección de código que se desea probar y la experiencia de la persona que desarrolla la prueba.

Algunos de los defectos no son detectados de manera temprana en las primeras fases de los proyectos, se detectan al llegar a las fases posteriores donde se realizan las pruebas de aceptación (tipo exploratorias y funcionales) como se aprecia en la figura No.1; lo cual implica realizar arreglos más costosos que pudieron detectarse en etapas más tempranas del desarrollo.



**Figura No.1.** Tipos de prueba vs peso ponderado en tiempo de desarrollo

**Fuente:** (Falgueras, 2003)

## 2. Planteamiento del problema

¿Cómo mejorar la curva de tiempo a través de la detección temprana de errores en la fase de codificación, con herramientas para la Generación Automática de Casos de Prueba aplicado a proyectos de desarrollo de *software*, en Dualbiz S.R.L.?

### 2.1 Objeto de Estudio

Herramientas para la Generación Automática de Casos de Prueba.

## **2.2 Campo de Acción**

Curva de tiempo a través de la detección temprana de errores en la fase de codificación, aplicado a proyectos de desarrollo de *software* en Dualbiz S.R.L.

## **3. Objetivos**

### **3.1 Objetivo General**

Evaluar la integración de una herramienta para la Generación Automática de Casos de Pruebas en la fase de codificación para la mejora de la curva de tiempo a través de la detección temprana de errores, aplicada a proyectos de desarrollo de *software*, en Dualbiz S.R.L.

### **3.2 Objetivos específicos**

1. Fundamentar teóricamente las herramientas para la Generación Automática de Casos de Prueba, que permita identificar las bondades de la herramienta, su uso, y requisitos necesarios en la implementación de la misma.
2. Diagnosticar la situación de la fase de codificación y la detección de errores como parte del proceso de desarrollo, para determinar los problemas actuales suscitados en los proyectos de *software* en Dualbiz S.R.L.
3. Aplicar la herramienta seleccionada para la Generación Automática de Casos de Prueba en la fase de codificación, a proyectos de desarrollo de *software*, para evaluar la mejora de la curva de tiempo a través de la detección temprana de errores, en Dualbiz S.R.L.

4. Validar el uso de herramientas para la Generación Automática de Casos de Prueba en la fase de codificación en la mejora de la curva de tiempo, aplicado a proyectos de desarrollo de *software* en Dualbiz S.R.L.

#### **4. Idea Científica a Defender**

Existe una mejora en la curva de tiempo a través de la detección temprana de errores en la fase de codificación de proyectos de desarrollo de *software* con la integración de herramientas para la Generación Automática de Casos de Prueba.

#### **5. Justificación**

Como se ha descrito anteriormente, los entregables en Dualbiz S.R.L. se ven afectados en retrasos, causando pérdidas económicas y de oportunidades en el mercado; la post venta de los productos desarrollados también tiene un número considerable de incidencias.

No se puede concluir que el retraso en el cronograma y también las incidencias encontradas en los productos, son causados por errores introducidos en la fase de codificación en su totalidad, pero si un porcentaje, el cual se desea mejorar.

En lo académico, se pueden observar un número creciente de trabajos relacionados a la incorporación de herramientas para la Generación Automática de Casos de Pruebas y el uso de las mismas; determinando un conjunto de condiciones o valores iniciales de un programa o elemento de *software* que determina la ejecución del mismo y permite comprobar el buen funcionamiento de dicho elemento de *software*.

En la industria no se ha podido constatar un número contundente de casos en ambientes reales, el presente trabajo de investigación plantea la incorporación de herramientas para la Generación Automática de Casos de Pruebas en la fase de codificación de proyectos de desarrollo de *software*, en Dualbiz S.R.L.

## **6. Delimitación de la investigación**

**Delimitación espacial:** La investigación se desarrollará en oficinas de la empresa Dualbiz S.R.L., que tiene localidad en Santa Cruz de la Sierra Bolivia, afectando la fase de codificación de los proyectos de desarrollo de *software*.

**Delimitación temporal:** El presente trabajo de investigación incorporará el uso de herramientas para la Generación Automática de Casos de Prueba a proyectos en actual desarrollo; se define como fecha inicial de integración de las herramientas propuestas agosto del 2017 y entrega final del análisis y propuesta para inicios de Marzo del 2018.

**Delimitación sustantiva:** Las herramientas propuestas para la investigación son *EvoSuite* e *IntelliTest* para la Automatización de los Casos de Pruebas, aplicadas a las tecnologías de uso presente en los proyectos de desarrollo de *software*, en Dualbiz S.R.L.

## **7. Diseño Metodológico**

### **7.1. Tipo de Investigación**

El presente trabajo de investigación, se enmarca dentro del esquema de investigación propositiva, siguiendo el lineamiento pre-experimental; en tanto se propone la incorporación de una herramienta para la Generación Automática de Casos

de Prueba a modo de mejorar la curva de tiempo en la fase de codificación, aplicados a proyectos de desarrollo de *software* en Dualbiz S.R.L.

## 7.2. Métodos de Investigación

- **Método Histórico Lógico:** Permitirá realizar un análisis cronológico de los referentes teóricos que caracterizan la Generación Automática de Casos de Prueba, a partir de analizar los antecedentes, el desarrollo y tendencias futuras.
- **Enfoque de sistemas:** Este método permitirá establecer y argumentar las relaciones entre las etapas que se definan durante la integración de la herramienta seleccionada para la Generación Automática de Casos de Pruebas en la fase de codificación, aplicados a proyectos de desarrollo de *software* en el contexto donde se investiga, con el fin de mejorar la curva de tiempo a partir de la detección temprana de errores en la fase mencionada.

## 7.3. Técnicas e Instrumentos de Investigación

### Técnicas:

- **Entrevista** a gerentes de Desarrollo y Operaciones para identificar algunos problemas actuales relacionados al objeto de estudio del trabajo de investigación.
- **Encuesta** a desarrolladores, relacionadas al campo de acción de la investigación, y poder relevar los problemas que inciden en el área de codificación desde una perspectiva interna.

Para la etapa de diagnóstico del estado actual del objeto de estudio y campo de acción, se utilizará el método de la Matriz de Vester a modo jerarquizar los

problemas para su posterior solución dentro de la propuesta al problema de investigación. Este método permitirá determinar la correlación de los problemas relevados y determinar problemas críticos, pasivos e irrelevantes que actúan sobre el campo de acción que a los efectos de la presente investigación es el tiempo de ejecución de los proyectos de desarrollo *software*, en Dualbiz S.R.L.

En el análisis estadístico de los resultados de la aplicación de los instrumentos de investigación en el diagnóstico se utilizarán herramientas de la estadística descriptiva y para la validación de la propuesta de solución al problema de investigación, el método T de *Student* propio de la estadística paramétrica.

#### **Instrumentos:**

- Cuestionario de entrevista a gerentes de Desarrollo y Operaciones (Anexo No.1).
- Cuestionario de encuesta a desarrolladores (Anexo No. 2).

#### **7.4. Población y Muestra**

En el caso de entrevistas y encuesta la población y muestra coincide en un ciento por ciento y está determinada por los gerentes y el equipo de desarrollo que realizará las pruebas.

## CAPÍTULO I. MARCO TEÓRICO Y CONCEPTUAL

En este capítulo, a partir del método histórico lógico, se desarrolla el sustento teórico necesario sobre el objeto de estudio de la investigación: “Herramientas para la Generación Automática de Casos de Pruebas”. Se revisa información relevante, de diferentes autores, transitando desde el concepto básico de las pruebas y luego se profundiza en cada uno de los conceptos de interés acerca de las herramientas posibles a utilizar en la propuesta; identificando las ventajas, el alcance, las especificaciones necesarias para la implementación y los cuidados a tener en consideración al momento de aplicar las mismas.

### 1.1. Acercamiento al concepto de pruebas

Es bien conocido que las pruebas son un componente esencial en el proceso de construcción de *software*, con el desarrollo de los lenguajes de cuarta generación (4GL) los cuales aceleran el proceso de implementación, la proporción de tiempo dedicado a las pruebas también se ven afectados significativamente. Así como también el mantenimiento y las actualizaciones de los sistemas actuales continúan creciendo, aun con métodos y técnicas de construcción y validación formales, los sistemas aún necesitan de las pruebas antes de su uso. Por consiguiente, la fase de pruebas es un componente importante de investigación en la generación de técnicas y procedimientos automatizados para la generación de las mismas. (Miller, 2012)

Paul Li (Luo, 1990) en su artículo “Técnicas de Pruebas de *Software* – Tecnología, maduración y estrategias de investigación”, realiza retrospectiva de cincuenta años de investigación de la técnica de prueba de *software*; examina la



maduración de la investigación de técnicas de prueba de *software* mediante el seguimiento de los principales resultados de investigación que han contribuido al crecimiento de esta área. También evalúa el cambio de paradigmas de investigación a lo largo del tiempo al rastrear los tipos de investigación, preguntas y estrategias utilizadas en varias etapas.

El concepto de prueba evolucionó junto con el de la definición y los objetivos a los cuales las pruebas de *software* se han dirigido con la investigación sobre pruebas y técnicas. A continuación, se revisa, brevemente, la evolución del concepto de pruebas utilizando el Modelo de Proceso de Prueba, propuesto por David Helperin (Helperin & Hetzel, 1998).

### **Fase I. Antes de 1956: Período orientado a la Depuración**

Las pruebas no se separaron de la depuración. En 1950, Turing escribió un famoso artículo, considerado el primero que analiza la prueba del programa. El artículo aborda la pregunta "¿Cómo sabríamos que un programa exhibe inteligencia?". De otra manera, si el requisito es construir dicho programa, entonces esta pregunta es un caso especial de "¿Cómo sabríamos que un programa satisface sus requisitos?". Turing definió una prueba operacional (requerimiento del comportamiento del programa) y un sistema de referencia (un ser humano) que fueran indistinguibles de un interrogador (probador). Esto pudiera considerarse la forma embrionaria de las pruebas funcionales. Los conceptos de la verificación del programa, la depuración y las pruebas no estaban claramente diferenciados para ese momento.

## **Fase II. 1957 – 78: Período orientado a la Demostración**

Prueba para asegurarse de que el *software* cumpla con su especificación. No fue sino hasta 1957 que se puso a prueba, lo que se llamó el pago del programa en ese momento, diferenciándose del período de depuración. En 1957, Charles Baker señaló que se consideraba que el "pago y envío del programa" tenía dos objetivos: "asegúrese de que el programa se ejecute" y "asegúrese de que el programa resuelva el problema". El último objetivo fue visto como el foco de las pruebas, ya que "asegurarse" a menudo se traducía en el objetivo de la prueba en satisfacer requisitos.

Como se analizó anteriormente, la depuración y las pruebas son en realidad dos fases diferentes. La distinción entre prueba y depuración descansaba en la definición de éxito. Durante este período, las definiciones enfatizan que el propósito de las pruebas es demostrar corrección: una prueba ideal, por lo tanto, tiene éxito solo cuando un programa no contiene errores. La década de 1970 también vio la idea generalizada de que el *software* se podía probar exhaustivamente. Esto llevó a considerar hacer el énfasis de la investigación en la prueba de cobertura del camino. Como se menciona en el artículo de Goodenough y Gerhart de 1975 "Pruebas exhaustivas definidas ya sea en términos de rutas de programas o en el dominio de entrada de un programa".

## **Fase III. 1979 – 82: Período orientado a la Destrucción**

Prueba para detectar fallas de implementación. En 1979, Myers escribió el libro "*The Art of Software Testing*", que proporcionó la base para un diseño más efectivo de la técnica de prueba. Por primera vez, las pruebas de *software* se describieron

“como el proceso de ejecutar un programa con la intención de encontrar errores” (Glenford, 1979). El punto importante fue que el valor de casos de prueba es mucho mayor si se encuentra un error.

Como en el período orientado a la demostración, se pudiera seleccionar inconscientemente los datos de prueba que tienen una baja probabilidad de causar fallas del programa. Si la prueba tiene la intención de mostrar que un programa tiene fallas, entonces los casos de prueba seleccionados tendrán una mayor probabilidad de detectarlos y la prueba es más exitosa. Este cambio en el énfasis condujo a la asociación temprana de pruebas y otras actividades de verificación/validación.

#### **Fase IV. 1983 – 87: Período orientado a la Evaluación**

El Instituto de Ciencias de la Computación y Tecnología de la Oficina Nacional de Normas, publicó la Directriz para la Validación, Verificación y Prueba del Ciclo de Vida del *Software* de Computadora en 1983 (citado por Lou), en la cual la metodología integra actividades de análisis, revisión y prueba para proporcionar evaluación de productos durante el ciclo de vida del *software*.

La guía, da la creencia de que un conjunto de técnicas para la Validación, Verificación y Pruebas (VV & T) pueden ayudar a asegurar el desarrollo y mantenimiento de *software* de calidad.

#### **Fase V. Desde 1988 a la actualidad: Período orientado a la Prevención**

En el libro *Software Testing Techniques* (Beizer, 1990), que contiene el catálogo más completo de Técnicas de Prueba, Beizer afirma que “El acto de diseñar pruebas es uno de los mecanismos más efectivos para prevenir errores”, de esta manera

extendió la definición de pruebas tanto a las actividades de detección como la prevención de errores. Esto condujo a una visión clásica del poder de las primeras pruebas.

En 1991, Hetzel dio la definición de que “las pruebas son planificación, diseño, construcción, mantenimiento y ejecución de pruebas y entornos de prueba”. Un año antes de esto, Beizer propuso cuatro ideas sobre las pruebas: 1. para demostrar que el *software* funcione; 2. para “romper” el *software*; 3. para reducir el riesgo; y 4. un estado de ánimo, es decir una preocupación total del ciclo de vida con la capacidad de prueba. Estas ideas llevaron pruebas de *software* para enfatizar en la importancia del diseño temprano de la prueba a lo largo del ciclo de vida del *software*.

El período orientado a la Prevención se distingue del orientado a la Evaluación, por el mecanismo, aunque ambos se centran en los requisitos y el diseño del *software* para evitar errores de implementación. El modelo de Prevención considera que la planificación de pruebas, el análisis de pruebas y las actividades de diseño de pruebas desempeñan un papel importante; mientras que el modelo de Evaluación se basa principalmente en técnicas de análisis y revisión distinta de las pruebas.

## **1.2. Pruebas de Software**

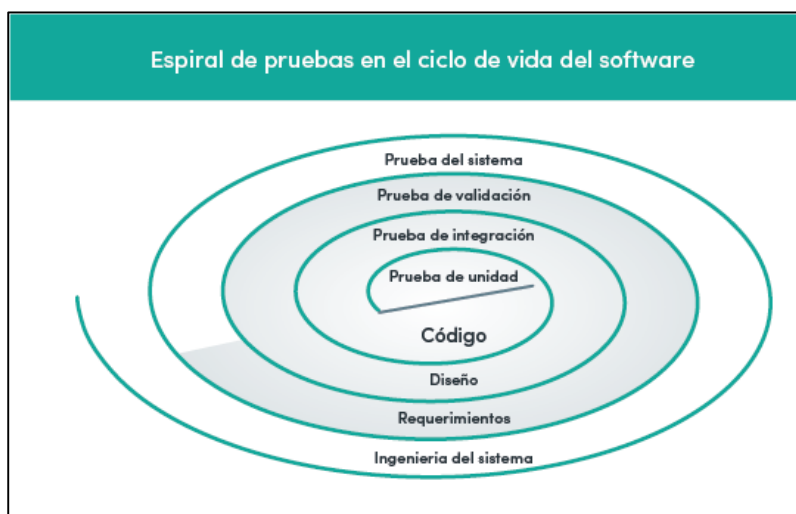
En diferentes publicaciones, la definición de prueba varía de acuerdo con el propósito, el proceso y el nivel de prueba descrita. Miller da una buena descripción de las pruebas en (Miller, 2012): “El objetivo general de las pruebas es afirmar la calidad de los sistemas de *software* mediante el ejercicio sistemático del *software* en circunstancias cuidadosamente controladas”.

La descripción de Miller de las pruebas ve la mayoría de las actividades de aseguramiento de la calidad del *software* como pruebas. Él sostiene que esa prueba debería tener la intención principal de encontrar errores. Una buena prueba es aquella que tiene una alta probabilidad de encontrar un error aún no descubierto y una prueba exitosa es aquella que descubre un error aún no descubierto. Esta categoría general de actividades de prueba de *software* se puede dividir aún más.

El proceso de *software* puede verse como la espiral que se ilustra en la figura No. 2, inicialmente la Ingeniería de Sistemas define el papel del *software* y conduce al análisis de los requerimientos del mismo, donde se establecen los criterios de dominio, función, comportamiento, desempeño, restricciones y validación de información para el *software*.

Al avanzar hacia adentro a lo largo de la espiral, se llega al diseño y finalmente a la codificación. Para desarrollar *software* de computadoras, se avanza en espiral hacia adentro (contra las manecillas del reloj) a lo largo de una línea que reduce el nivel de abstracción en cada vuelta.

Una estrategia para probar el *software* también puede verse en el contexto de la espiral de la figura No. 2. La prueba de unidad comienza en el vértice de la espiral y se concentra en cada unidad (por ejemplo, componente, clase o un objeto de contenido de una *webapp*) del *software*, cómo se implementó en el código fuente.



**Figura No.2.** Espiral de pruebas en el ciclo de vida del *software*

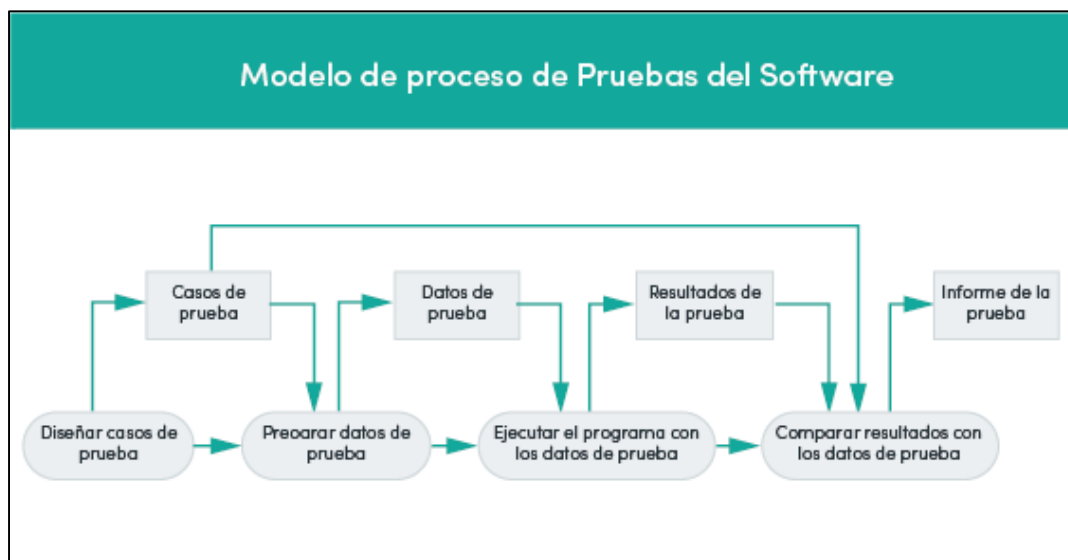
**Fuente:** (Sommerville, 2012) Capítulo 17.1 Estrategia de las Pruebas

La prueba avanza al moverse hacia afuera a lo largo de la espiral, hacia la prueba de integración, donde el enfoque se centra en el diseño y la construcción de la arquitectura del *software*. Al dar otra vuelta hacia afuera de la espiral, se encuentra la prueba de validación, donde los requerimientos establecidos como parte de su modelado, se validan confrontándose con el *software* que se construyó.

Finalmente, se llega a la prueba del sistema, donde el *software* y otros elementos del sistema se prueban como un todo. Para probar el *software* de cómputo, se avanza en espiral hacia afuera en dirección de las manecillas del reloj a lo largo de líneas que ensanchan el alcance de las pruebas con cada vuelta.

La actividad de probar el *software* en sí misma es un componente esencial en cualquier proyecto de desarrollo de *software*, una prueba básica puede consistir desde generar una entrada simple a un programa y revisar que el resultado propuesto esté de acuerdo a los requerimientos establecidos para dicho programa,

hasta la implementación de casos de pruebas para distintas fases del proyecto, según se muestra en la figura No. 3.



**Figura No.3.** Un Modelo de Proceso de Pruebas

**Fuente:** (Sommerville, 2012) Cap.7 Pruebas del *Software*

La prueba es un conjunto de actividades que pueden planearse por adelantado y realizarse de manera sistemática. Por esta razón, durante el proceso de *software*, debe definirse una plantilla para la prueba del *software*: un conjunto de pasos que incluyen métodos de prueba y técnicas de diseño de casos de prueba específicos.

Autores como Somerville, han propuesto algunas estrategias de prueba de *software*; todas proporcionan una plantilla para la prueba y tienen las siguientes características genéricas:

- Para realizar una prueba efectiva, debe realizar revisiones técnicas efectivas. Al hacerlo, eliminará muchos errores antes de comenzar la prueba.
- La prueba comienza en los componentes y opera “hacia afuera”, hacia la integración de todo el sistema de cómputo.

- Diferentes técnicas de prueba son adecuadas para distintos enfoques de Ingeniería de *Software* y en diferentes momentos en el tiempo.
- Las pruebas son realizadas por el desarrollador del *software* y (para proyectos grandes) un grupo de prueba independiente.

Prueba y depuración son actividades diferentes, pero la depuración debe incluirse en cualquier estrategia de prueba.

### 1.3. El espectro de las pruebas

Existe un conjunto de pruebas que se utilizan en el ciclo de vida del *software*, entre las que se puede detallar:

**Pruebas unitarias (*Unit Testing*)**, se realiza en el nivel más bajo. Prueba la unidad básica de *software*, que es la más pequeña pieza de *software* comprobable, a menudo se denomina "unidad", "módulo" o "componente", indistintamente.

**Pruebas de integración (*Integration Testing*)**, se realizan cuando dos o más unidades probadas se combinan en una estructura más grande.

**Pruebas de sistema (*System Testing*)**, tienden a afirmar la calidad de extremo a extremo de todo el sistema. La prueba del sistema a menudo se basa en la especificación funcional/requisito del sistema. También son verificados los atributos de calidad no funcionales, tales como: confiabilidad, seguridad y mantenibilidad.

**Pruebas de Aceptación**, se realiza cuando el sistema completo se transfiere desde los desarrolladores al cliente o usuario. El propósito de las pruebas de aceptación es más bien dar confianza de que el sistema está trabajando, para encontrar errores.



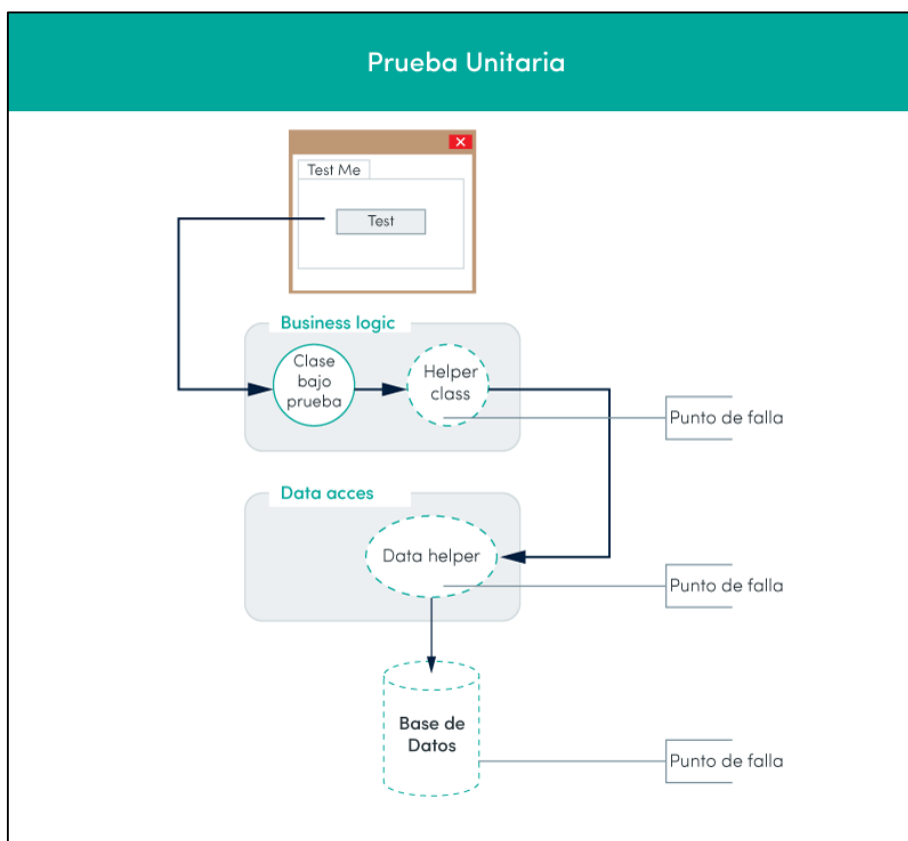
**Análisis Estático**, el análisis estático se centra en el rango de métodos que se utilizan para determinar o estimar la calidad del *software* sin referencia a las ejecuciones reales. Las técnicas en esta área incluyen la inspección del código, el programa análisis, análisis simbólico y verificación de modelos.

**Análisis dinámico**, trata con métodos específicos para determinar y/o aproximar la calidad del *software* a través de ejecuciones reales, es decir, con datos reales y en circunstancias reales (o simuladas). Las técnicas en esta área incluyen la síntesis de insumos, el uso de procedimientos de prueba dictados estructuralmente y la automatización de la generación del entorno de prueba.

#### 1.4. Prueba Unitaria

Roy Osheroov en su libro “*The Art of Unit Testing*” brinda una definición sencilla de un *Caso de Prueba*, como “código desarrollado para probar código”, parece algo redundante pero en efecto así lo es, es decir un caso de prueba es un programa que va a evaluar de manera asertiva los resultados de otro programa.

También se menciona una definición clásica de Caso de Prueba como: “Una porción de código (usualmente un método) que involucra otra porción de código y verifica la exactitud de algunas suposiciones, después de todo si la suposición resulta ser errónea, la prueba unitaria falla.” (Osheroov, 2009), según se observa en la figura No. 4.



**Figura No.4.** Ilustración de una prueba básica

**Fuente:** (Osherov, 2009) Cap. 2. Pruebas unitarias

El número de casos de prueba necesarios para probar un programa de *software* es infinito, por lo que es imposible conseguir un programa totalmente probado. Además, el proceso de prueba es costoso y puede suponer el 50% del coste total del desarrollo de *software* (Beizer, 1990).

Por estos motivos, las técnicas para la Generación Automática de Casos de Prueba tratan de encontrar, de forma eficiente, un conjunto pequeño de casos de prueba que permitan cumplir un determinado criterio de suficiencia. Debido a estos factores, muchos desarrolladores optan por no desarrollar pruebas unitarias dejando la carga de pruebas a fases posteriores de pruebas de validación o pruebas de sistema (Osherov, 2009).

En la figura No. 5, a continuación, se puede apreciar un ejemplo escrito en c# para una clase “Suma” con un solo método, el método “sumar” cuyo resultado es la suma de los dos números enteros, los mismos se reciben en el constructor de la clase.

```
public class Suma {  
    private int num1;  
    private int num2;  
  
    public Suma (int n1, int n2) {  
        num1 = n1;  
        num2 = n2;  
    }  
    public int sumar() {  
        int resultado = num1 + num2;  
        return resultado;  
    }  
}
```

**Figura No.5.** Clase a probar Escrita en C#

**Fuente.** Elaboración propia

Para la clase “Suma” definida en la figura anterior, se describe a continuación, en la figura No. 6, un conjunto de casos de pruebas definida en la Clase de Prueba (*TestClass*) “*SumaTest*”; la misma está desarrollada en .NET con lenguaje c# y el Framework *MStest* de *Microsoft*, donde se puede observar que presenta dos métodos de prueba (*TestMethod*): el método “SumaPositivos” que prueba la suma de dos números positivos y el método “SumaNegativos” que prueba la suma de dos números negativos; para ambos casos de prueba se tiene una aserción de tipo verdad para el resultado correcto.

```

[TestClass()]
public class SumaTest

[TestMethod()]
public void SumaPositivos()
{
    Suma S = new Suma(2,3);
    Assert.IsTrue(S.Sumar()== 5);
}

[TestMethod()]
public void SumaNegativos()
{
    Suma S = new Suma(2,-3);
    Assert.IsTrue(S.Sumar()== -1);
}

```

**Figura No.6.** Prueba unitaria Realizada en .NET c# con *MSTest Framework*

**Fuente.** Elaboración propia

En la siguiente figura No. 7, se presenta el mismo ejemplo para una clase escrita en *Java*, con la diferencia en la notación de los atributos de la clase perteneciente al *framework* de pruebas *Junit*.

```

public class SumaTest {
    @Test
    public void sumaPositivos () {
        System.out.println ("Sumando dos números positivos ...");
        Suma S = new Suma(2, 3);
        assertTrue (S.sumar() == 5);
    }
    @Test
    public void sumaNegativos () {
        System.out.println ("Sumando dos números negativos ...");
        Suma S = new Suma (-2, -3);
        assertTrue (S.sumar() == -5);
    }
    @Test
    public void sumaPositivoNegativo () {
        System.out.println("Sumando un número positivo y un número negativo ...");
        Suma S = new Suma (2, -3);
    }
}

```

**Figura No.7.** Prueba unitaria escrita con *Junit Framework*

**Fuente.** Elaboración propia

Del ejemplo anterior, se comprueba lo expuesto por Osharov, el número de casos de prueba puede ser infinito, para el caso de la clase “Suma” se realizan dos Casos de Pruebas: una para números positivos y otro para números negativos, asumiendo un conjunto de números reales como entrada para la función Suma.

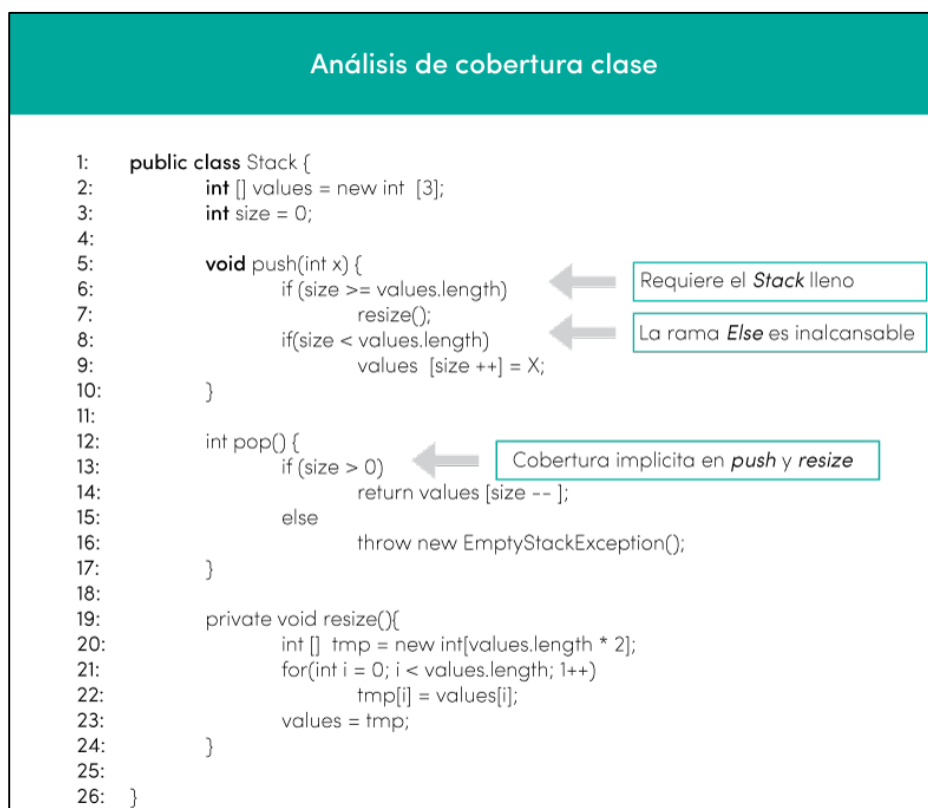
### 1.5. Generación Automática de Casos de Prueba

Como se ha descrito anteriormente, un enfoque común, en la literatura contemporánea, es la Generación de Casos de Prueba para cobertura de código en base a la meta de la técnica; es decir que, dependiendo de la técnica utilizada para la generación automática se debería cumplir la meta establecida. (Ej. Número de ramas, en criterio de cobertura de ramas).

Sin embargo, sin importar la técnica utilizada, el resultado de una *Suite de Test* es difícil de predecir, a medida de que cada caso de prueba para una meta puede implícitamente cubrir una porción de código de otro número de metas futuras de cobertura.

Esto, comúnmente, es conocido como “cobertura fortuita” (Gorden Fraser, 2011), para la Clase <Stack> de la figura No. 8, se puede realizar el siguiente análisis con lo expuesto anteriormente. El ejemplo de la clase <Stack> extraído de (Fraser & Arcuri, 2013) resalta los siguientes puntos con respecto a la cobertura: para la línea 5 la cobertura de la rama *false* es más difícil de cubrir que la rama *true*, debido a que para llegar a cubrir la rama *false* se debería tener el *array* de la pila lleno. También resalta que algunos puntos son imposibles debido a que no tendría una entrada que pueda cubrir la rama, específicamente hablando de la línea 7.

En este punto, concuerda con otros autores como Allen Goldberg hablando de Coberturas y caminos posibles. Aun cuando en este caso específico es fácil de detectar la rama de cobertura imposible; la pregunta final es, ¿cuánto esfuerzo y presupuesto se tiene para llegar a la meta de cobertura?.



**Figura No.8.** Ejemplo de análisis de cobertura clase <Stack>

**Fuente:** (Fraser & Arcuri, 2013)

Actualmente existen métricas aceptadas para determinar estos puntos aunque siga siendo una pregunta abierta; completando el escenario, en la línea 11 centra la atención a una cobertura dependiente, puesto que no es posible llegar a cubrir esta rama sin antes cubrir la línea 7 del *push* y el *false* de la línea 5 del mismo método. Definiendo de manera singular el caso anterior, se puede determinar que, llegar a

una meta de cobertura, en algunos casos, puede ser complicado dependiendo de la meta establecida o por lo menos llegar a un porcentaje aceptado métricamente.

El diseño de Casos de Prueba generado automáticamente, según lo expuesto anteriormente, es una meta difícil de cumplir, pero esto da origen a un concepto medianamente nuevo denominado Pruebas Evolutivas (*Evolutionary Testing*) (Sebastian Bauersfeld, 2011), el mismo es caracterizado por el uso de técnicas meta-heurísticas de Búsqueda y Algoritmos Combinados para la generación de los casos de pruebas.

Para la clase *Stack* del ejemplo anterior, en la figura No. 9 se puede apreciar los Casos de Prueba generados por *EvoSuite*. Esto, precisamente, debido a la no linealidad del código escrito, es decir sentencias de *loops*, *if-then* que normalmente son el resultado complejo de búsqueda de espacios no lineales y discontinuos. En la figura No. 9 se puede observar la *Suite* de Pruebas generadas para la clase *<Stack>* que consiste en dos Casos de Pruebas, uno para el caso de la pila vacía y otro para la ejecución normal de la pila, con una cantidad X de datos introducidos.

Suite de Pruebas utilizado para el análisis de la cobertura de la clase pila	
1:   Stack stack0 =new Stack();	Stack stack0 =new Stack();
2: <b>try</b> {	int int0 = -510;
3:         stack.pop();	stack0.push(int0);
4:   } <b>catch</b> (EmptyStackException e) {	stack0.push(int0);
5:   }	stack0.push(int0);
6:	stack0.push(int0);
7:	stack0.pop();

**Figura No.9.** Suite de Pruebas utilizado para el análisis de la cobertura de la clase pila

**Fuente:** (Fraser & Arcuri, 2013)

De lo descrito anteriormente, se debe tomar en cuenta las siguientes medidas de complejidad para tener una idea más exacta de la lógica de esta generación automatizada: (EOLC) *Executable lines of code*, (HALL) *Halstead's Length Vocabulary*, (CYC) *Cyclomatic Complexity*, (MY) *Myers Interval*, (NLC) *Nesting Level Complexity*, (NTA) *Number of test Aims*.

- **Líneas de Código Ejecutable (*EOLC Executable Lines Of Code*)**, se determinan las líneas de código ejecutable, es decir, las sentencias de flujo de información, anotaciones y propiedad de flujo de control no son examinadas, solamente las líneas que contienen sentencias ejecutables.
- **Vocabulario y longitud de Haldstead (*HALL Halstead's Length Vocabulary*)**, esta medida está basada en el conteo de operandos y operadores; así como también, el número de veces que son usados en el *software* a examinar.
- **Complejidad Ciclomática (*CYC Cyclomatic Complexity*)**, es definida como el número de vértices del grafo de flujo de control del programa a examinar, menos el número de nodos, más dos veces el número de los componentes unidos.
- **Intervalo de Myers, (*MY Myers Interval*)** es una extensión de la complejidad ciclomática, que toma la complejidad de los elementos de las ramas de condición para tener una cuenta más exacta. El valor de esta métrica es básicamente la suma de los operadores lógicos AND y OR en las expresiones condicionales a investigar.



- **Nivel de Complejidad de Anidamiento (*NLC Nesting level complexity*)**

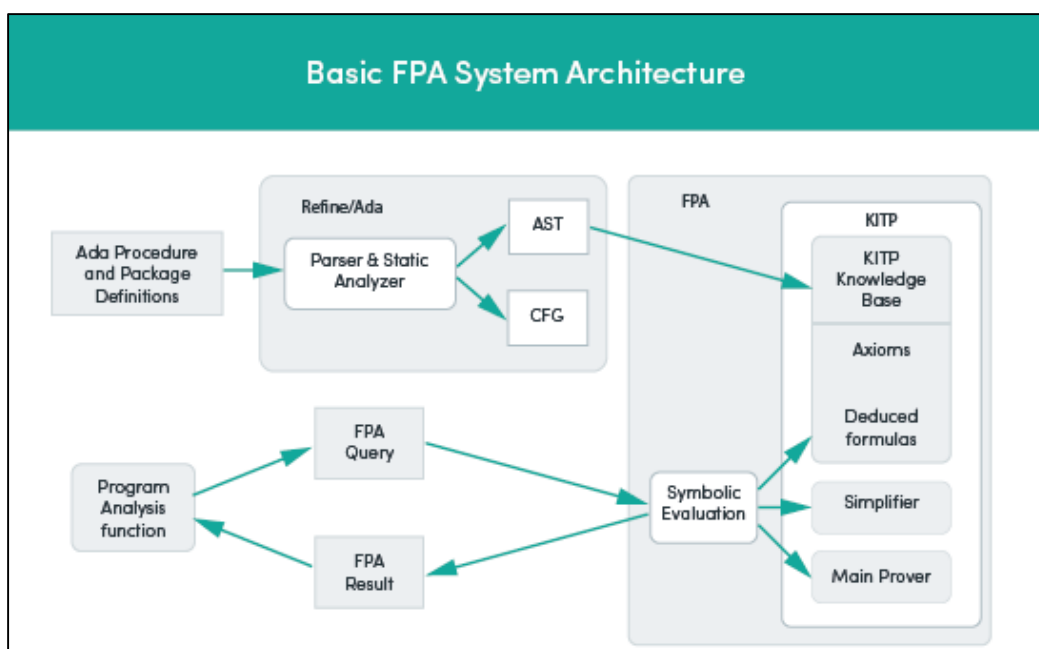
Evalúa la complejidad del *software* con respecto al nivel de anidamiento de sus condiciones y declaraciones.

- **Número de Objetivos de la Prueba (*NTA Number of Test Aims*)** El objetivo de la Prueba Evolutiva es encontrar un *set* de *data* para la prueba del objeto a probar por lo menos alguna vez. En otras palabras, puede ser usado como significado del esfuerzo de probar.

### 1.5.1. Análisis de caminos posibles

Dado un camino de un flujo de control, un camino posible es posible, si existe una asignación de valores de entrada, (es decir, variables globales, parámetros, entre otros) que conducen la ejecución por el camino. A esto, es posible un conjunto de controles de trayectorias, si uno de sus miembros es factible. Se pudiera entonces utilizar una expresión regular para describir un conjunto de caminos, tal expresión se llama ruta de expresión regular (**PRE**) (Allen Goldberg, 1994), en la siguiente figura No. 10 se puede observar el conjunto de componentes en la ejecución simbólica.

Dentro del análisis de caminos posibles se puede determinar: “Dado un procedimiento y tipo asociado, constantes y variables. Una ruta de expresión regular **P** desde un inicio **S** hasta un punto final **E**, Una fórmula **@**; el flujo de control intenta determinar si hay un cálculo de estado, es decir, valores para variables de programa en **S** que impulsa la ejecución de los caminos denotada por **P**, de tal forma que la fórmula evaluada **@** en el punto **S** es verdadera.



**Figura No.10.** FPA Básico de la Ejecución Simbólica

**Fuente:** (Allen Goldberg, 1994)

Usualmente **S** es la entrada al procedimiento y se buscan los valores necesarios globales para la ejecución con los valores requeridos, **FPA** construye una formula lógica en primer orden que satisface si y solo si hay entradas cumpliendo la especificación. A esto se denomina “Evaluación Simbólica” a través de la construcción de fórmulas y axiomas.

La tarea de la Evaluación Simbólica es construir fórmulas lógicas de primer orden para pasar al teorema de pruebas. Algunas de estas fórmulas pueden ser de tipo Datos, Axiomas derivados de las declaraciones de tipo y otras expresiones derivadas; usualmente condiciones de control dentro del código del procedimiento, a diferencia de las variables, algunas de las cuales dependen de un almacén de valores. Para hacer frente a esta diferencia, cada programa variable se representa como una familia lógica variable. La familia esta indexada por el programa donde los valores pueden ser diferentes; la evaluación simbólica “hacia atrás” se utiliza para

generar aseveraciones que relacionan las variables en diferentes puntos del programa. (Allen Goldberg, 1994)

### 1.5.2. Ingeniería basada en búsqueda

Las técnicas de algoritmos basados en búsqueda han sido ampliamente utilizadas en la generación de *data* para las pruebas, desde los años 70s con la introducción de algunos conceptos como distancia de ramas (*Branch Distance*), Búsqueda Dispersa (*Scatter Search*) y otros (Glover & Laguna, 1977).

La búsqueda dispersa 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.

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.

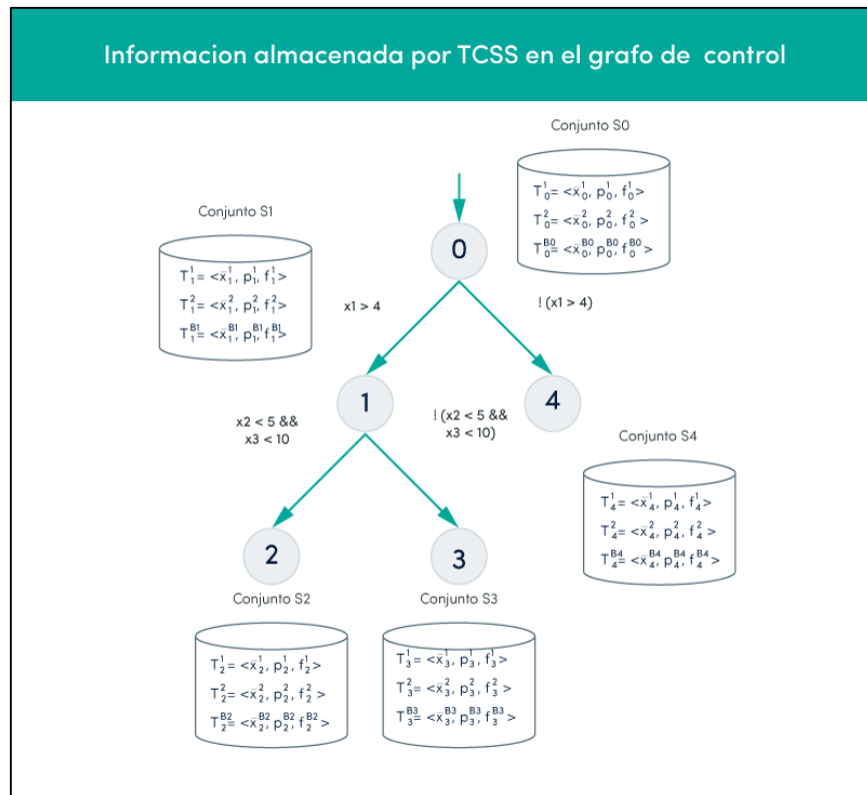
El generador de Casos de Prueba desarrollado basado en Búsqueda Dispersa se denomina TCSS (*Test Coverage Scatter Search*) y utiliza el grafo de control de flujo asociado al programa bajo prueba, para almacenar información relevante durante el proceso de búsqueda de casos de prueba. Con este grafo es posible determinar qué ramas han sido cubiertas debido a que el programa bajo prueba ha sido instrumentado para determinar el camino seguido por cada caso de prueba ejecutado en él (Blanco, Diaz, & Tuya, 2006).

El objetivo de TCSS es generar Casos de Prueba que permitan cubrir todas las ramas de un programa. Este objetivo se puede dividir en sub-objetivos, consistiendo cada uno de ellos en encontrar Casos de Prueba que alcancen un determinado nodo del grafo de control de flujo.

Para alcanzar estos sub-objetivos, TCSS trabaja con un conjunto de soluciones (conjunto de referencia) en cada nodo del grafo de control de flujo. Cada uno de estos conjuntos de soluciones se denomina  $S_k$  (donde  $k$  es el número de nodo) y contiene varios elementos  $T_k^c \rightarrow \langle \bar{x}_k^c, P_k^c, f_k^c \rangle$ , donde  $\bar{x}_k^c$  es una solución (un caso de prueba) que alcanza el nodo  $k$ ,  $P_k^c$  es el camino recorrido por la solución y  $f_k^c$  es la distancia que indica lo cerca que dicha solución está de pasar por su nodo hermano, es decir, el nodo cuya decisión de entrada es la negación de la decisión del nodo  $k$ . Cada una de las soluciones almacenadas en un nodo  $k$  está compuesta por

los valores de las variables de entrada que hacen ciertas tanto las decisiones de entrada a los nodos precedentes al nodo  $k$  en un determinado camino que permite llegar a él, como la del propio nodo  $k$ .

La estructura del grafo de control de flujo asociado a un programa, junto con la información que se almacena en cada nodo se puede ver en la figura No. 11.



**Figura No.11.** Información almacenada por TCSS en Grafo de control

**Fuente:** (Allen Goldberg, 1994)

El conjunto de soluciones de un nodo  $k$ ,  $S_k$  tiene un tamaño máximo  $B_k$ . Este tamaño es distinto para cada nodo  $k$  y depende de la complejidad del código situado por debajo de dicho nodo  $k$ , que puede ser medida por medio de la complejidad ciclomática del grafo de control de flujo resultante, de tomar como raíz al propio nodo  $k$ . Este valor de complejidad es multiplicado por un factor fijo para disponer de una

cantidad razonable de soluciones en cada conjunto  $S_k$  que permita generar otras nuevas.

TCSS tratará de hacer los conjuntos  $S_k$  lo más diversos posibles, utilizando una función de diversidad para generar soluciones que puedan cubrir distintas ramas del programa. El objetivo de TCSS es obtener la máxima cobertura de ramas, por lo que deben encontrarse soluciones que permitan cubrir todos los nodos del grafo de control de flujo. Como estas soluciones se almacenan en los nodos, el objetivo de TCSS es, por tanto, que todos los nodos tengan al menos un elemento en su conjunto  $S_k$ .

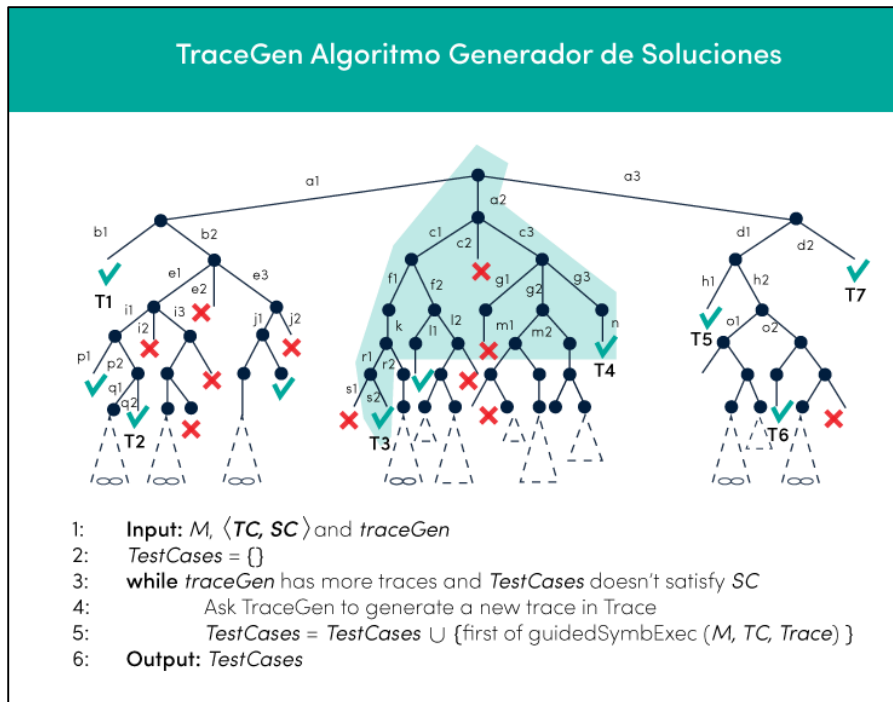
### 1.5.3. Cálculo de la distancia asociada a una solución

Para que una solución alcance un determinado nodo, debe cumplirse la decisión de entrada al mismo. Por ello, si se desea calcular lo cerca que la solución está de pasar por el nodo hermano se debe utilizar la decisión de entrada a dicho nodo hermano. (Blanco, Diaz, & Tuya, 2006) Por ejemplo, en la figura No.11, la solución  $\bar{x}_1^1$  alcanza el nodo 1 y el valor de distancia  $f_1^1$  se calcula utilizando la decisión de entrada al nodo 4 ( $!(x_1 > 4)$ ).

Para el cálculo de la distancia  $f_k^c$ ; en primer lugar se evalúa cada condición que forma parte de la decisión de entrada al nodo hermano, según los valores de las variables de entrada que constituyen la solución. Posteriormente, se calcula el valor  $f_k^c$  de la decisión. Si en la decisión intervienen operadores *AND*, la distancia  $f_k^c$  será el resultado de sumar la evaluación de las condiciones falsas, pues son las que impiden que se alcance al nodo hermano.

Si en la decisión intervienen operadores *OR*, la distancia  $f_k^c$  será el valor mínimo de las evaluaciones de las condiciones, ya que todas ellas son falsas y con que se cumpla una sola, se alcanzaría el nodo hermano. Si la decisión está negada, simplemente se aplican las leyes de D'Morgan.

**Generación de nuevas soluciones**, en cada iteración del algoritmo, TCSS selecciona un nodo para generar las nuevas soluciones en el proceso de búsqueda por medio de la combinación de las soluciones almacenadas en su conjunto  $S_k$ . Los criterios que rigen la selección del nodo con el que trabajar, de modo general, se observan en la figura No. 12, árbol de decisión para el algoritmo TCSS.



**Figura No.12.** *TraceGen* algoritmo Generador de Soluciones

**Fuente:** (Blanco, Diaz, & Tuya, 2006)

TCSS selecciona un número constante  $b$  de elementos  $T_k^c \rightarrow \langle \bar{x}_k^c, P_k^c, f_k^c \rangle$  del conjunto  $S_k$  que previamente no hayan sido utilizados, para generar nuevas soluciones, intentado que esas soluciones  $(\bar{x}_k^c)$  cubran caminos distintos  $(P_k^c)$  y

tengan menor valor de distancia ( $f_k^c$ ). Con las soluciones seleccionadas se forman todos los posibles pares  $(\bar{x}_k^j, P_k^h)$ , con  $j \neq h$ , y a partir de cada uno de ellos se generan cuatro nuevas soluciones como resultado de aplicar las siguientes combinaciones, elemento a elemento.

Cada nueva solución es examinada para determinar si los valores de sus variables de entrada se encuentran situados dentro del rango que cada una de ellas puede tomar. De no ser así, se debe aplicar sobre dicha solución el método de mejora, que se limita a modificar los valores de las variables para que no sea considerada inválida. Con estas combinaciones se pretende generar soluciones que se alejen de las originales y soluciones que se sitúen entre ellas. Además, con los criterios de selección de las soluciones se intentan combinar soluciones diversas (recorren distintos caminos) que estén próximas a producir un salto de rama.

Si el nodo seleccionado para generar nuevas soluciones no posee al menos dos soluciones que puedan ser empleadas para realizar las combinaciones, se lleva a cabo un proceso de *backtracking*. El programa bajo prueba es ejecutado con cada nueva solución. Cada una de estas ejecuciones puede causar la actualización de los conjuntos  $S_k$  de los nodos alcanzados.

### **1.6. IntelliTest**

*IntelliTest*, es una herramienta de *Microsoft* para la Generación Automática de Casos de Pruebas que existe embebido en el *IDE (Integrated Development Enviroment) Visual Studio .NET*. Básicamente sobre lo expuesto anteriormente



explora el código .NET para generar datos de prueba y un conjunto de pruebas unitarias.

Para cada instrucción en el código, se genera una entrada de prueba que ejecutará esa instrucción. Se lleva a cabo un análisis de caso para cada bifurcación condicional en el código. Por ejemplo, se analizan las instrucciones *if*, las aserciones y todas las operaciones que pueden producir excepciones. Con este análisis puede generar los datos de pruebas que deben usarse en una prueba unitaria parametrizada para cada método. También crea pruebas unitarias con una cobertura de código elevada.

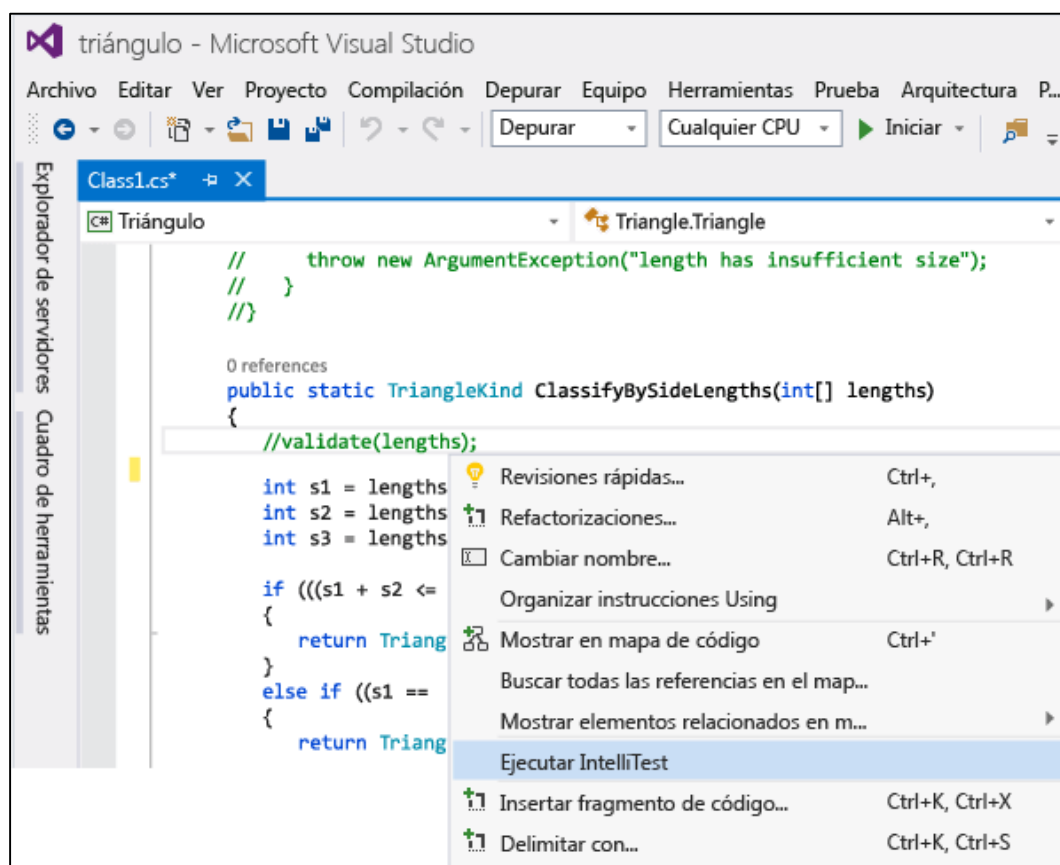
Cuando se ejecuta *IntelliTest*, se puede ver fácilmente qué pruebas son las que fallan y agregar cualquier código para corregirlas. Se puede seleccionar las pruebas generadas que quiere guardar en un proyecto de prueba para proporcionar un conjunto de regresión. Cuando cambie el código, se vuelve a ejecutar *IntelliTest* para mantener sincronizadas las pruebas generadas con los cambios de código.

## **Disponibilidad y extensiones**

Los comandos de menú Crear *IntelliTest* y Ejecutar *IntelliTest*, solo están disponibles en la edición *Enterprise* de *Visual Studio* 2015 y versiones posteriores.

- Solo admiten código de C# que tenga como destino .NET Framework.
- Son extensibles y admiten la emisión de pruebas en formato *MSTest*, *MSTest V2*, *NUnit* y *xUnit*.
- No admiten la configuración x64.

Para generar las pruebas unitarias, los tipos deben ser públicos. De lo contrario, debe crear primero la solución con la plantilla de pruebas. Se abre la solución de trabajo en *Visual Studio* y luego abre el archivo de clase que tiene los métodos que desea probar. En la figura No. 13 se puede observar que, basta un clic con el botón derecho en un método del código y generar pruebas unitarias para el código del método, haciendo clic en “Ejecutar *IntelliTest*” para tener un primer resultado.

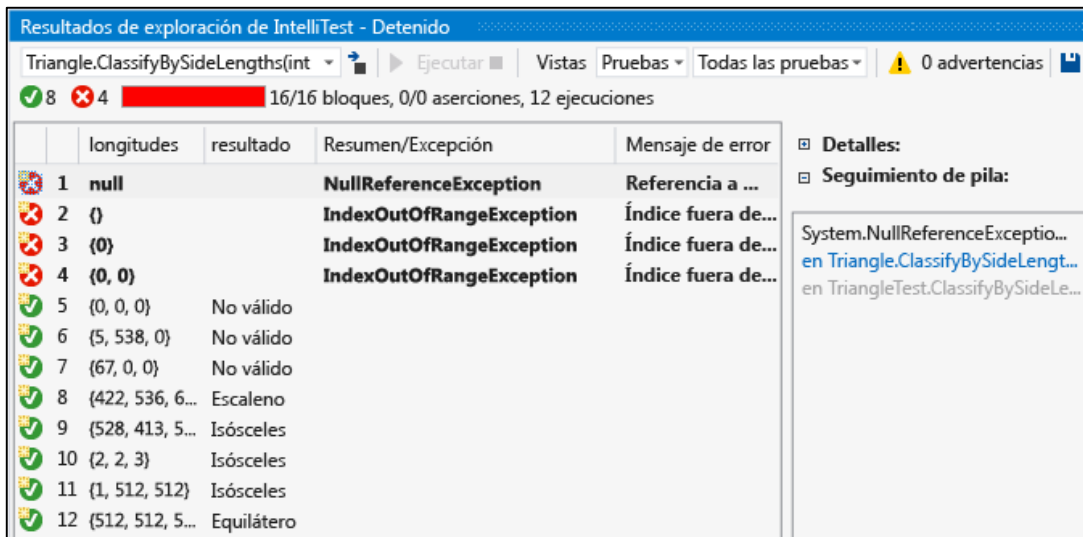


**Figura No.13.** Ejecutar *IntelliTest* (Generación Automática de Pruebas)

**Fuente:** (Warren & Saisang, 2015)

*IntelliTest* ejecuta el código muchas veces con diferentes entradas. Cada ejecución se representa en la tabla que muestra los datos de las pruebas entrantes y la salida o excepción resultante, como se puede apreciar en la siguiente figura No. 14, en la generación de pruebas unitarias para todos los métodos públicos en una

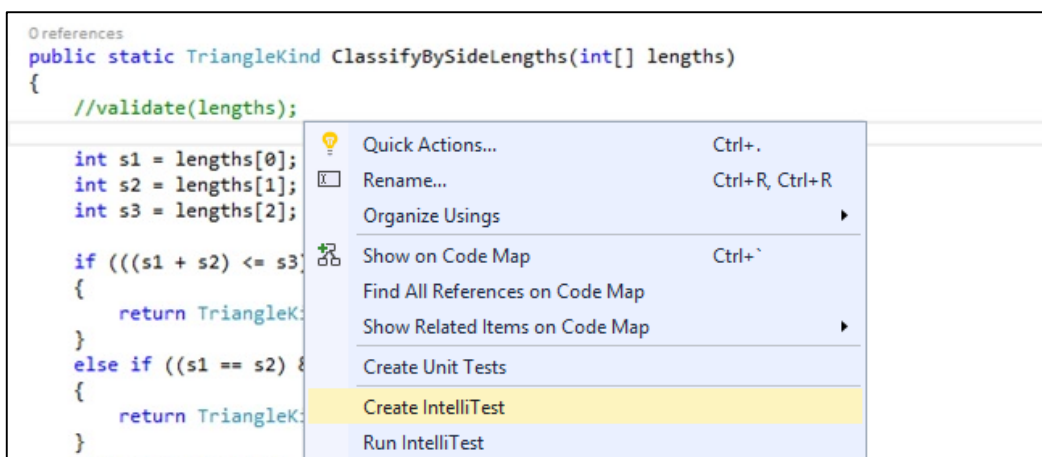
clase, simplemente se necesita hacer clic con el botón secundario en la clase, en el lugar del método específico.



**Figura No.14.** Resultados de Exploración IntelliTest

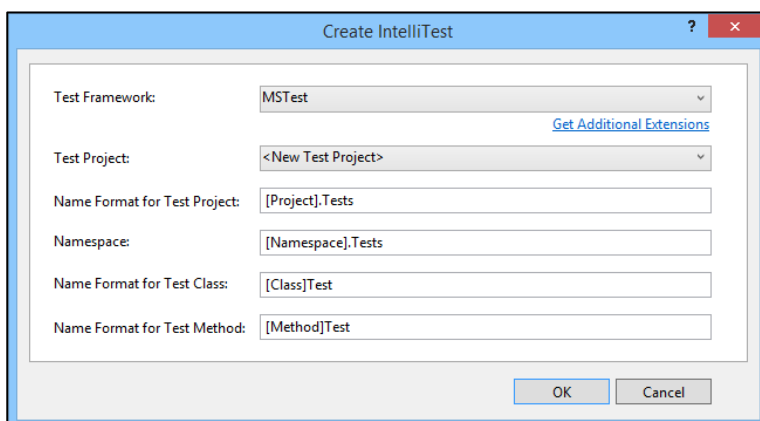
**Fuente:** (Warren & Saisang, 2015)

A continuación del paso “Ejecutar *IntelliTest*”; aparece la lista desplegable en la ventana Resultados de exploración, Figuras No. 15 y 16 respectivamente; luego es posible visualizar las pruebas unitarias y los datos entrantes para cada método en la clase dentro de un nuevo proyecto de Pruebas.



**Figura No.15.** Ejecución *IntelliTest* para una Clase completa

**Fuente:** (Warren & Saisang, 2015)



**Figura No.16.** Menú emergente Creación de Proyecto de Pruebas *IntelliTest*

**Fuente:** (Warren & Saisang, 2015)

En cuanto a las pruebas superadas, se puede comprobar que los resultados de los cuales se informa en la columna de resultados coincidan con sus expectativas con respecto a su código. Si algunas pruebas generan un error, se puede corregir el código según corresponda. Después se vuelve a ejecutar *IntelliTest* para validar las correcciones y así se puede continuar el proceso para llegar a una validación correcta de los resultados esperados.

Una vez realizado el estudio de los fundamentos que sustentan teóricamente el objeto de estudio, se está en condiciones de realizar el diagnóstico del estado actual del campo de acción, como parte que se transforma durante la investigación, en el contexto que se investiga, Dualbiz S.R.L.

## CAPITULO II - DIAGNÓSTICO

El presente capítulo tiene por objetivo determinar la situación actual en Dualbiz S.R.L. a través de la aplicación de instrumentos de investigación como la entrevista y la encuesta. Se realiza una valoración de la situación de actual de la fase de codificación y la detección de errores como parte del proceso de desarrollo. Se utiliza la Matriz de Vester con los problemas recolectados de las encuestas y entrevistas; para de esta manera especificar la asociación de los mismos; es decir generar grupo de problemas críticos, pasivos, indiferentes y los activos con un enfoque cualitativo y cuantitativo.

### 2.1. Acercamiento al contexto que se investiga

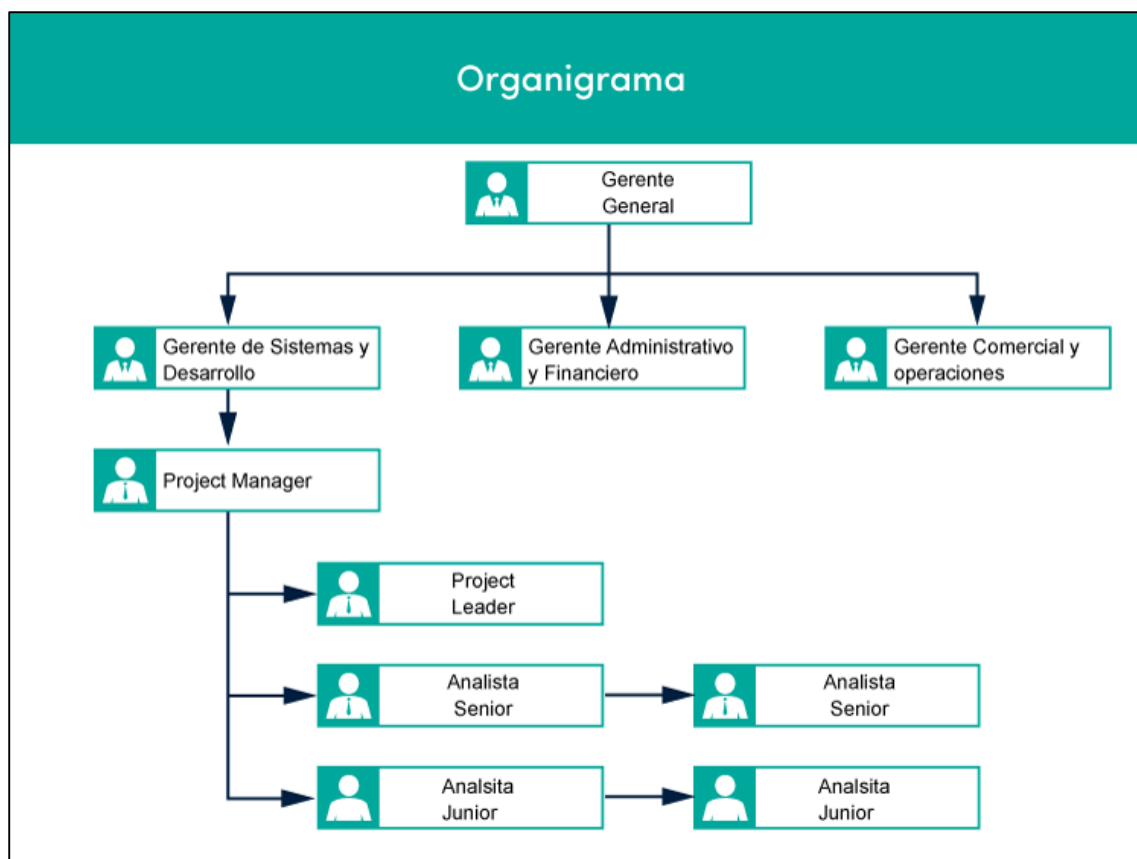
Dualbiz S.R.L. es una empresa privada del rubro de Tecnología de la Información, que se dedica al desarrollo de *software*. Tiene como objetivo: brindar a sus clientes soluciones informáticas prácticas, sencillas y confiables, que faciliten el trabajo diario y les permitan obtener mayor eficiencia y productividad. La oficina central se encuentra ubicada en la ciudad de Santa Cruz y está conformada por 13 profesionales con diferentes especialidades.

Ofrece una gama de productos para tecnología *web* y cuenta con personal especializado en el desarrollo de aplicaciones para dispositivos móviles, considerando los avances tecnológicos y las tendencias actuales.

#### 2.1.1. Estructura organizacional de la empresa

La empresa tiene una estructura organizacional establecida, con funciones, responsabilidades y líneas de mando conocidas por todos los funcionarios. Sin

embargo, no cuenta con manual de funciones ni procedimientos formales, hecho que, en ocasiones, deriva en fallas del personal en ciertas etapas del proceso de desarrollo de *software*. La siguiente figura No. 17, muestra la estructura organizacional de la empresa.



**Figura No.17.** Estructura Organizacional Dualbiz S.R.L.

**Fuente:** Elaboración propia

### 2.1.2. Equipo de Desarrollo

Actualmente la empresa comercializa soluciones en Tecnologías *.Net*, *Java JSP*, y *Android* para lo cual dispone de 13 personas involucradas en Equipos de Desarrollo, como se muestra en la figura No.18. Cada equipo, dependiendo de la tecnología, está integrado por 4 ó 5 desarrolladores; siendo el equipo de *.NET* con

más integrantes, debido a los productos que comercializa y la cantidad de clientes y cambios necesarios para los mismos.



**Figura No.18.** Equipos de Desarrollo por Tecnología, Dualbiz S.R.L.

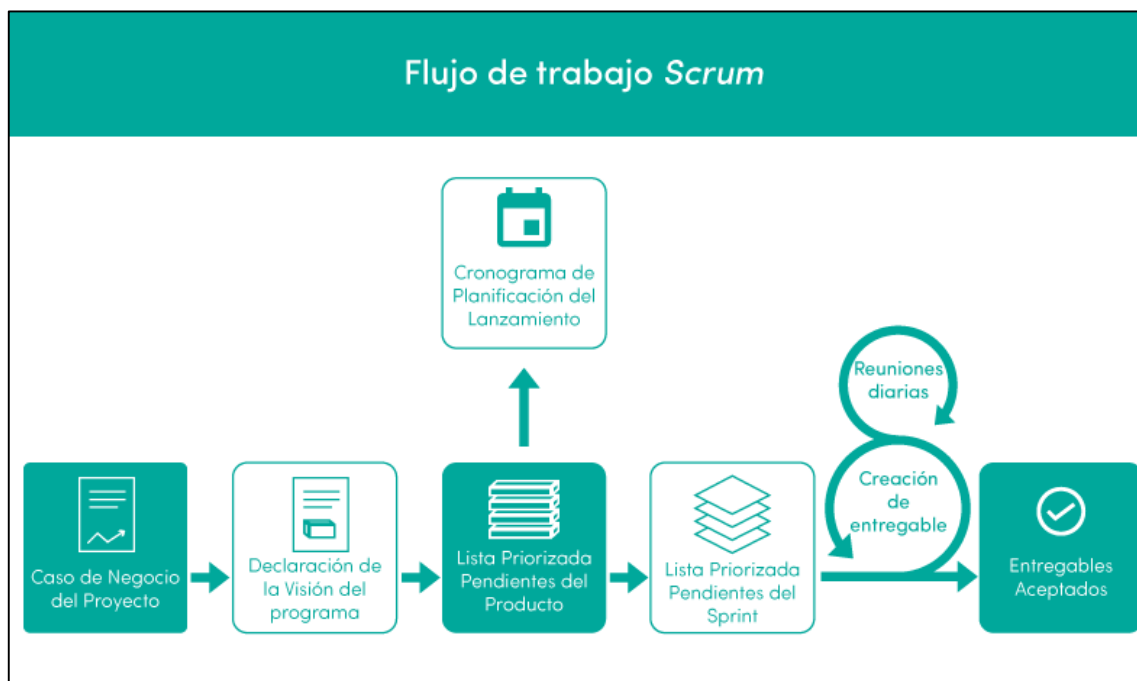
**Fuente:** Elaboración propia

Para la organización y mantenimiento de los proyectos se tiene un Sistema de Gestión de Configuración *Microsoft Team Foundation System*; TFS, integrado a su gestión documental en *Sharepoint*, donde se encuentran alojados los elementos y requerimientos necesarios para realizar el trabajo cotidiano en la empresa.

### 2.1.3. Fase de codificación y pruebas

Los proyectos de Desarrollo de *Software* son gestionados siguiendo algunos lineamientos del marco de trabajo *Scrum*, es decir, se realiza un relevamiento de los requerimientos en una primera etapa, figura No.19; para luego definirlos y llevarlos a una pila central y dividirlos en *Sprint*, con una duración máxima de dos semanas;

proporcionando al finalizar el proceso, entregables de valor al cliente final (Satpathy & Sutherland, 2016). Todo esto queda plasmado en un Documento de Visión y Alcance (DVA) que contiene algunos elementos como la Matriz de Interesados, Plantilla de Requerimientos y la Tabla Cruzada como requerimientos finales.



**Figura No.19.** Flujo de trabajo *Scrum*

**Fuente:** (Satpathy & Sutherland, 2016; Northrop, 2011)

Estos requerimientos se transforman en las historias de usuario para realizar la ejecución de las mismas dentro del esquema y cronograma propuesto por el documento mencionado anteriormente y las estimaciones realizadas.

### Pruebas Internas

Durante la fase de codificación, los requerimientos son desarrollados por el equipo, el mismo que realiza las pruebas de entrega; es decir, se realizan las pruebas necesarias antes de dar como finalizado el requerimiento en cuestión.



## Validación y Aceptación

Una vez se tiene un entregable para el cliente, al finalizar un *Sprint*, se genera una versión prueba en un ambiente donde se notifica al cliente que puede revisar los cambios pactados. En esta etapa se espera no encontrar o encontrar un mínimo de errores para dar por terminado los requerimientos del cliente final.

### 2.2. Procedimiento para el Diagnóstico

Para la elaboración del diagnóstico; se utiliza como instrumentos de investigación inicial, la entrevista y la encuesta, así como también se emplean datos históricos del Sistema de Gestión de Configuración, para obtener información relevante a las preguntas de los instrumentos mencionados. Es decir, obtener información cuantitativa de estadísticas de errores, incidencias pasadas de algunos proyectos utilizados como referencia en la presente investigación.

Con la información obtenida, de las fuentes anteriormente mencionadas, se determina utilizar la Matriz de Vester o matriz de motricidad de problemas.

La Matriz de Vester, fue creada por el químico y economista alemán Frederic Vester, y ha sido utilizada con acierto en diferentes campos que incluyen consultoría técnica y administrativa, estrategias de negocio, planeación, educación e investigación científica; recibiendo con este modelo el premio "*Philipp Morris Research Award*".

Es una herramienta que permite medir la relación causa – efecto y organiza los problemas en línea de motricidad, es decir, identifica cuáles problemas son el resultado de una serie de procesos o procedimientos incorrectos (efectos) y cuáles

son dinámicos porque su estructura e importancia los posiciona como el lugar clave que genera efectos a otros departamentos, áreas o productos y por eso se identifican como problemas causales, de mayor relevancia o poderosos (Northrop, 2011).

Para la elaboración de la Matriz de Vester se deben seguir un conjunto de pasos:

1. **Identificar los problemas:** Se identifican los problemas; se puede utilizar cualquier técnica de investigación para la recolección de estos.
2. **Elaboración de Ficha Técnica:** Se elabora una ficha técnica para cada uno de los problemas identificados desde los instrumentos aplicados, la cual contiene un resumen del problema, su descripción, tendencia y fuente de datos. De esta manera, se sustenta la situación problemática.
3. **Construcción de la Matriz (dependencia e influencia):** Se realiza el análisis según ponderación de cada problema, luego se realiza una triangulación de los problemas que aparecen en las filas y contrastando con los problemas que aparecen en las columnas. A partir de la siguiente pregunta: ¿Cómo influye el problema 1 sobre el problema 2? Y así sucesivamente, hasta completar cada una de las casillas, proporcionando un valor según lo ponderado. Esto determina los valores de X, Y para la posterior construcción del plano cartesiano.
4. **Construcción del Plano Cartesiano:** Una vez obtenido los valores de los ejes X, Y; se realiza el cálculo de promedio de influencia y dependencia. Luego se elabora la gráfica del plano cartesiano con sus respectivos cuadrantes. Finalmente, se ubican los problemas en el cuadrante correspondiente, según se observa en la figura No.20.



**Figura No.20.** Cuadrante resultante de problemas Matriz de Vester

**Fuente:** (Northrop, 2011)

**5. Clasificación de los problemas:** se clasifican los problemas según la ubicación en el cuadrante de la figura anterior, obteniendo la siguiente clasificación:

- **Cuadrante Poder (problemas críticos o centrales):** Este cuadrante representa la más alta motricidad o causalidad. Los problemas que aparezcan ubicados en este sector son los más importantes y la metodología afirma que si estos obstáculos son resueltos de forma eminente, los otros problemas se resolverán.
- **Cuadrante de Conflicto (problemas activos o causas):** Este cuadrante se denomina de esta forma porque en él se ubican problemas con alta y mediana motricidad, los problemas que se alejan del vértice hacia el extremo superior se consideran altos y los que se posicionan en el inferior tienen motricidad media.

- **Cuadrante de Indiferencia (problemas pasivos o efectos):** Los problemas que se ubiquen en este cuadrante son de baja motricidad o problemas efecto, por eso se recomienda no tomar éstos para resolverlos porque en su condición de consecuencia no ayudaría a resolver las situaciones.
- **Cuadrante de Inercia (problemas indiferentes):** Este cuadrante es de nulidad, es decir que los problemas que aparecen en este cuadrante deben descartarse totalmente porque éstos no tienen conexión o relevancia ni con aquellos que son causales, ni con los que son efecto.

### 2.2.1. Diseño de la Encuesta y Entrevista

**Diseño de la Entrevista:** Para definir el contenido de la entrevista se elabora el diseño basado en el proceso de operacionalización (Anexo No. 3) a las variables identificadas, Codificación y Casos de Prueba; obteniendo como resultado el cuadro No. 1, que define propósito, enfoque, perspectiva y contexto.

**Cuadro No.1.** Entrevista a Gerencia de Proyectos Dualbiz S.R.L.

Objeto de Estudio	Estado actual de la fase de codificación
<b>Propósito</b>	Relevar las causas o problemas asociados a la fase de codificación y las incidencias en los productos de comercialización, en Dualbiz S.R.L.
<b>Enfoque Cualitativo</b>	Efectividad, defectos encontrados, personas en equipo de desarrollo, presencia de problemas en fase de codificación.
<b>Perspectiva</b>	Gerente de Proyectos, Gerente de Marketing (enfoque de arriba hacia abajo).
<b>Contexto</b>	Este estudio se realiza con personal gerencial recibiendo un enfoque de arriba hacia abajo de los problemas identificados en Dualbiz S.R.L. de manera general.

**Fuente:** Elaboración propia

A partir de este instrumento, se obtiene un enfoque de arriba hacia abajo de los problemas recolectados por la encuesta, así como también, se refinan los problemas

específicos del campo de acción sobre los que puede actuar el objeto de estudio, es decir, las herramientas para la Generación Automática de Casos de Prueba

**Diseño de la encuesta:** De la misma forma, se realiza el diseño de la encuesta como se observa en el cuadro No. 2, definiendo: propósito, enfoque, perspectiva y contexto; de esta manera se evidencian problemas detectados en la entrevista y recolectan otros problemas relacionados con el objeto de estudio.

**Cuadro No.2.** Diseño de la Encuesta a Equipo de Desarrollo Dualbiz S.R.L.

Objeto de Estudio	Problemas de la fase de codificación y Generación de Casos de Pruebas
<b>Propósito</b>	Contrastar los datos obtenidos de la entrevista sobre las causas o problemas asociados a la fase de codificación y las incidencias en los productos de comercialización, en Dualbiz S.R.L. Conocer aspectos relativos a la Generación de Casos de Pruebas.
<b>Enfoque Cuantitativo</b>	Nivel de conocimiento sobre pruebas unitarias, errores, problemas asociados a fase de codificación.
<b>Perspectiva</b>	Gerente de Proyectos, Gerente de Marketing y Desarrolladores.
<b>Contexto</b>	Este estudio se realiza con Personal de la empresa recibiendo los distintos enfoques y la percepción de problemas identificados en Dualbiz S.R.L.

**Fuente:** Elaboración propia

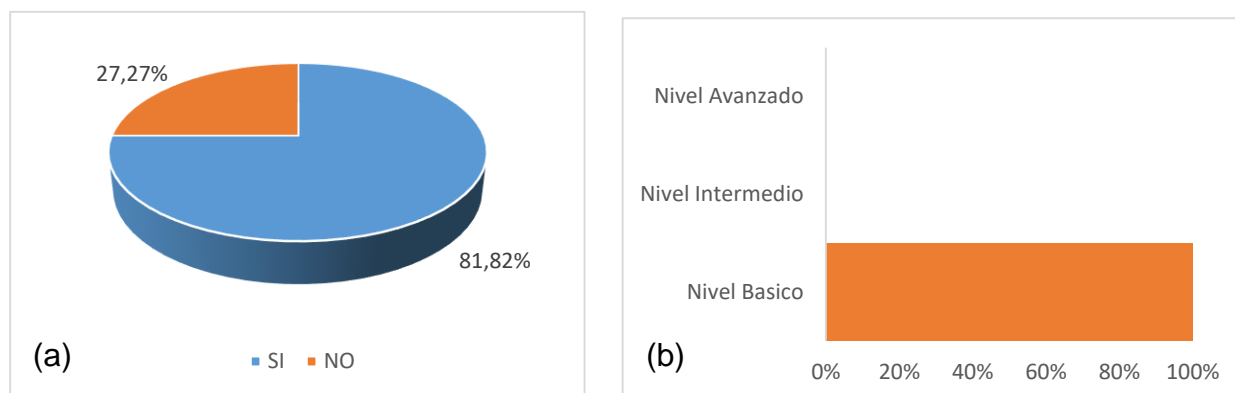
En esencia, se desea encontrar problemas inmersos dentro del proceso de codificación; teniendo el enfoque interno de los problemas que atañen a la manera tradicional de trabajar dentro de la empresa. Como resultado se quiere determinar una estadística sobre qué problemas parecen ser comunes en la percepción de grupo total de trabajo

### **Análisis de los resultados.**

Se aplica la encuesta (Anexo No. 2) al grupo de desarrollo en Dualbiz S.R.L.; un total de 13 personas. El resultado de la misma, se contrasta con los datos obtenidos

de la entrevista, (Anexo No.1), aplicada a los Gerentes de Área. Se detallan, a continuación, algunos de los resultados obtenidos sobre el estado actual de la Generación Automática de Casos de Prueba y otros problemas detectados.

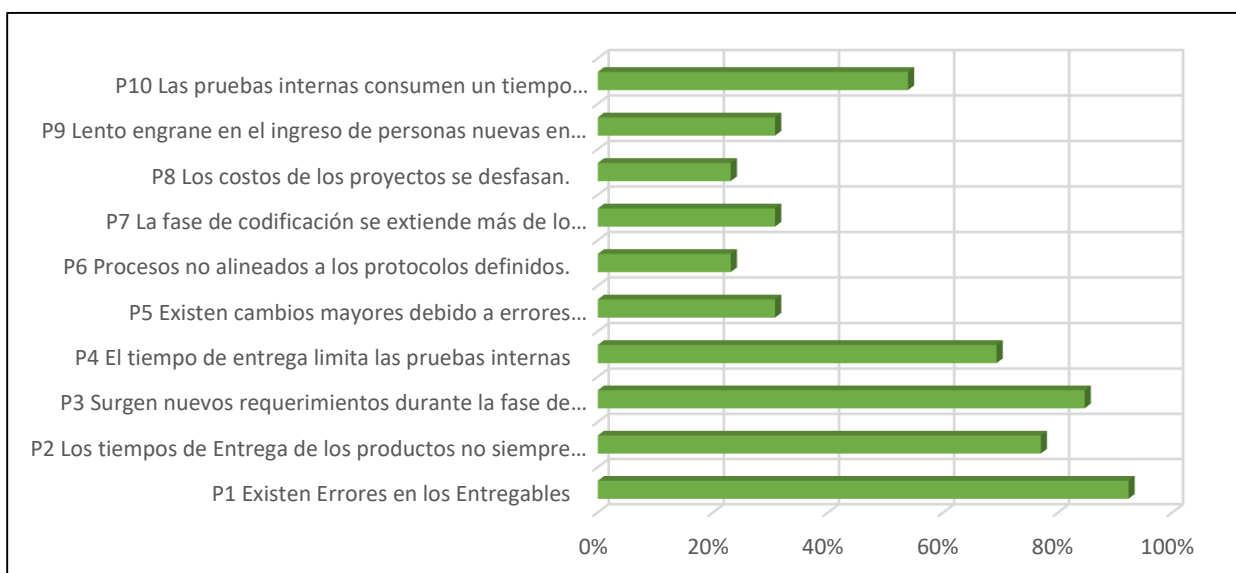
Acerca del conocimiento sobre Casos de Prueba, el 27,3% del grupo de desarrollo, conoce o ha realizado pruebas unitarias. Del porcentaje que responde positivo, el 100% tiene un conocimiento básico de la Herramienta, según se muestra en la figura No. 21. En la Entrevista a los gerentes sobre este punto, se detalla que actualmente no existen casos pruebas asociados a los proyectos.



**Figura No.21.** (a) Conocimiento de Casos de Prueba. (b) Nivel de Conocimiento acerca de Casos de Prueba

**Fuente:** Elaboración propia

Con respecto a la fase de codificación, se logra recolectar una cantidad de problemas asociados directamente al Área de Desarrollo, en Dualbiz S.R.L. con una puntuación relacionada al número de personas que identifica el mismo problema. Un resumen de estos problemas se aprecia en la figura No. 22.



**Figura No.22.** Ponderación de los problemas relevados en la Encuesta y Entrevista

**Fuente:** Elaboración propia

### 2.2.2. Elaboración de Matriz de Vester

Aplicadas las encuestas y entrevistas, con el análisis de los resultados se procede a la definición de la Matriz de Vester según cuadro No. 3, para determinar los problemas actuales referentes a la implementación de proyectos en fase de codificación y puntualmente sobre el equipo de Desarrollo.

**Cuadro No.3** Diseño Matriz de Vester, caso Dualbiz S.R.L.

Preguntas	Respuestas
¿Qué problema serán analizados?	Problemas asociados a la fase de codificación y las incidencias post-puesta en marcha en los productos de comercialización en Dualbiz S.R.L.
¿Dónde ocurren estos problemas?	Ocurren en los productos desarrollados en Dualbiz S.R.L.
¿A qué o a quiénes afectan?	De manera directa a los tiempos de entrega de productos desarrollados en Dualbiz y a los clientes finales.
Tema	Problemas asociados a la fase de codificación que dilatan el proceso de implementación en Dualbiz S.R.L.

**Fuente:** Elaboración propia

A continuación se detallan los pasos del procedimiento de Diagnóstico.

## 1. Identificación de problemas

Posterior al relevamiento de los problemas detectados en la encuesta y la entrevista; los mismos se refinan y seleccionan en base a los criterios del diseño de Matriz Vester propuesto en el cuadro anterior; es decir, para los problemas asociados a la fase de codificación de los proyectos de desarrollo de software, en Dualbiz S.R.L. y de esta manera obtener la matriz de problemas descrita a continuación, en el cuadro No.4.

**Cuadro No.4.** Matriz refinada de problemas para elaboración de Matriz de Vester

Cod.	Problemas
<b>P1</b>	Existen errores en los entregables
<b>P2</b>	Los tiempos de entrega de los productos no siempre se respetan
<b>P3</b>	Surgen nuevos requerimientos durante la fase de codificación
<b>P4</b>	El tiempo de entrega limita las pruebas internas
<b>P5</b>	Existen cambios mayores debido a errores detectados en fases finales del proyecto
<b>P6</b>	Procesos no alineados a los protocolos definidos.
<b>P7</b>	La fase de codificación se extiende más de lo planificado.
<b>P8</b>	Los costos de los proyectos se desfasan.
<b>P9</b>	Lento engrane en el ingreso de personas nuevas en los proyectos.
<b>P10</b>	Las pruebas internas consumen un tiempo considerable en la codificación

**Fuente:** Elaboración propia



## 2. Ficha Técnica de los Problemas

Del Cuadro No.5 al 14 se obtiene la especificación de cada uno de los problemas propuestos en la Matriz refinada de Problemas. Dicha especificación contiene un enunciado del mismo, una descripción breve, la tendencia y su fuente origen, para con esto realizar el análisis de los mismos.

**Cuadro No.5.** Ficha técnica P1

Detalle	Ficha Técnica P1
<b>Enunciado del problema</b>	Existen errores en los entregables
<b>Descripción</b>	Luego de la puesta en marcha existen errores detectados por el cliente.
<b>Tendencia</b>	Minimizar la presencia de los mismos
<b>Fuente de Datos</b>	Entrevista y encuesta

**Fuente:** Elaboración propia

**Cuadro No.6.** Ficha Técnica P2

Detalle	Ficha Técnica P2
<b>Enunciado del problema</b>	Los tiempos de entrega de los productos no siempre se respetan
<b>Descripción</b>	Los tiempos pactados por contrato con el cliente no siempre se respetan
<b>Tendencia</b>	Eliminar este desfase
<b>Fuente de Datos</b>	Entrevista

**Fuente:** Elaboración propia

**Cuadro No.7.** Ficha Técnica P3

Detalle	Ficha Técnica P3
<b>Enunciado del problema</b>	Surgen nuevos requerimientos durante la fase de codificación
<b>Descripción</b>	Durante la fase de codificación muchas veces se detectan requerimientos no revisados con anterioridad o se introducen nuevos requerimientos no pactados con el cliente
<b>Tendencia</b>	No se han revisado elementos para salvar esta contingencia.
<b>Fuente de Datos</b>	Entrevista y encuesta

**Fuente:** Elaboración propia

**Cuadro No. 8.** Ficha Técnica P4

Detalle	Ficha Técnica P4
<b>Enunciado del problema</b>	El tiempo de entrega limita las pruebas internas
<b>Descripción</b>	Los desarrolladores se quejan que es difícil medir la complejidad de las pruebas internas y no se estiman bien las tareas.
<b>Tendencia</b>	Minimizar
<b>Fuente de Datos</b>	Entrevista y encuesta

**Fuente:** Elaboración propia

**Cuadro No.9.** Ficha Técnica P5

Detalle	Ficha Técnica P5
<b>Enunciado del problema</b>	Existen cambios mayores debido a errores detectados en fases finales del proyecto
<b>Descripción</b>	Los errores detectados en etapas finales al proyecto muchas veces generan un retraso, puesto que deben volver a codificarse y probarse en conjunto con otros programas dependientes.
<b>Tendencia</b>	No se han revisado elementos para salvar esta contingencia.
<b>Fuente de Datos</b>	Entrevista y encuesta

**Fuente:** Elaboración propia

**Cuadro No.10.** Ficha Técnica P6

Detalle	Ficha Técnica P6
<b>Enunciado del problema</b>	Procesos no alineados a los protocolos definidos.
<b>Descripción</b>	Los desarrolladores no siguen el proceso de pruebas para liberar un entregable al cliente final o para realizar la publicación de los cambios.
<b>Tendencia</b>	Minimizar
<b>Fuente de Datos</b>	Entrevista.

**Fuente:** Elaboración propia

**Cuadro No.11.** Ficha Técnica P7

Detalle	Ficha Técnica P7
<b>Enunciado del problema</b>	La fase de codificación se extiende más de lo planificado.
<b>Descripción</b>	Los desarrollos pueden o no estar bien cuantificados, lo que provoca extender la fase de codificación por un tiempo mayor al definido.
<b>Tendencia</b>	Revisar mejor
<b>Fuente de Datos</b>	Entrevista y encuesta

**Fuente:** Elaboración propia

**Cuadro No.12.** Ficha Técnica P8

Detalle	Ficha Técnica P8
<b>Enunciado del problema</b>	Los costos de los proyectos se desfasan.
<b>Descripción</b>	Algunos proyectos involucran una cantidad de personalizaciones pactadas en el contrato del cliente, las cuales normalmente se extienden y afectan en tiempo de entrega y por ende en los costos.
<b>Tendencia</b>	Minimizar
<b>Fuente de Datos</b>	Entrevista

**Fuente:** Elaboración propia

**Cuadro No.13.** Ficha Técnica P9

Detalle	Ficha Técnica P9
<b>Enunciado del problema</b>	Lento engrane en el ingreso de personas nuevas en los proyectos.
<b>Descripción</b>	Cuando se contrata personal nuevo en el equipo de desarrollo, se tiene una curva de adaptación alta debido a los procedimientos, estándar de codificación, realización de pruebas internas y otros.
<b>Tendencia</b>	Se tienen manuales con algunos procedimientos y estándar, no se ha resuelto acciones para mejorar la inducción de gente al equipo.
<b>Fuente de Datos</b>	Encuesta

**Fuente:** Elaboración propia

**Cuadro No.14.** Ficha Técnica P10

Detalle	Ficha Técnica P10
<b>Enunciado del problema</b>	Las pruebas internas consumen un tiempo considerable en la codificación
<b>Descripción</b>	Los desarrolladores creen que las pruebas internas pueden ser un proceso largo y no son fáciles de estimar en el desarrollo de proyectos nuevos.
<b>Tendencia</b>	No se conoce la tendencia
<b>Fuente de Datos</b>	Encuesta

**Fuente:** Elaboración propia

### 3. Construcción de Matriz (dependencia e Influencia)

Para la elaboración de la matriz de dependencia e influencia; siguiendo los pasos propuestos por Vester, se debe resolver la correlación de cada uno de los problemas P1 a P10 cruzado por cada uno de los problemas P1 a P10; revisando el

nivel de influencia y teniendo el cuidado que un problema no se puede contrastar contra sí mismo.

La ponderación de la Matriz se lleva a cabo con la valoración propuesta del cuadro No.15, resolviendo la pregunta siguiente, por ejemplo: para P1: “Existen Errores en los entregables”, ¿cómo influye P1, sobre P2: “Los tiempos de entrega de los productos no se respetan”? y se le da una valoración de 0; es decir no existe influencia de P1 “que tengamos errores en los entregables” con P2. Y así, sucesivamente, se continúa el análisis con cada uno de los problemas.

**Cuadro No.15.** Relación de Influencia para Matriz Vester

Valor	Descripción
<b>0</b>	No existe relación directa entre el primer y el segundo problema.
<b>1</b>	Existe una influencia débil entre el primer y el segundo problema.
<b>2</b>	Existe una influencia mediana entre el primer y el segundo problema.
<b>3</b>	Existe una influencia fuerte entre el primer y segundo problema

**Fuente:** Elaboración propia

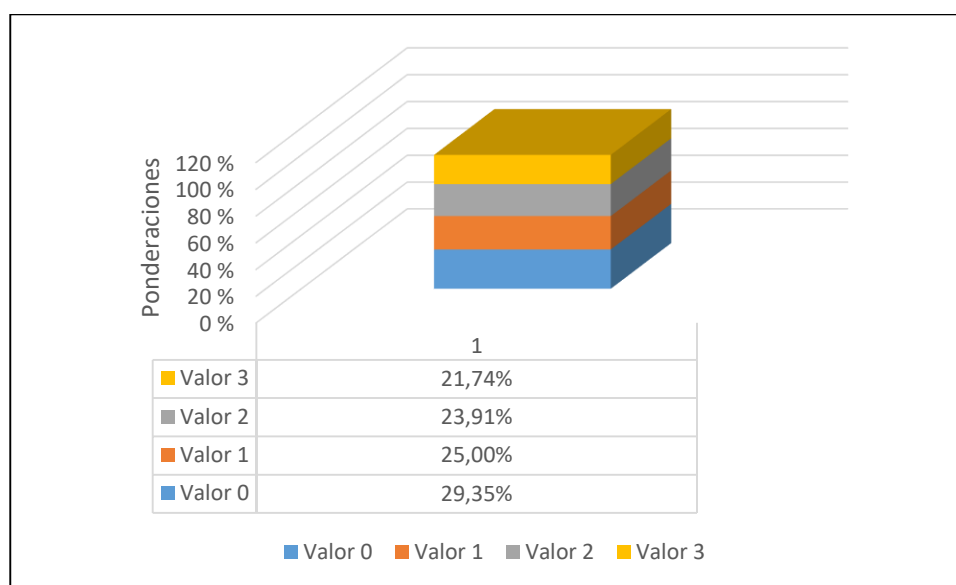
A continuación, en el cuadro No.16, se tiene la matriz de problemas, resuelta con la ponderación correspondiente a cada uno de los problemas detectados en la fase de codificación, para posteriormente realizar el análisis de estas ponderaciones; determinando su consistencia y la correlación entre cada uno de los problemas.

**Cuadro No.16.** Matriz de Dependencia e Influencia

	Descripción del Problema	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	X
<b>P1</b>	Existen errores en los entregables	0	0	1	3	1	0	1	0	0	3	9
<b>P2</b>	Los tiempos de entrega de los productos no siempre se respetan	1	0	0	2	0	1	2	3	0	2	11
<b>P3</b>	Surgen nuevos requerimientos durante la fase de codificación	1	3	0	3	2	0	3	2	0	0	14
<b>P4</b>	El tiempo de entrega limita las pruebas internas	2	2	0	0	0	0	0	0	0	3	7
<b>P5</b>	Existen cambios mayores debido a errores detectados en fases finales del proyecto	2	3	0	0	0	0	3	2	0	0	10
<b>P6</b>	Procesos no alineados a los protocolos definidos.	2	1	1	0	2	0	2	0	2	1	11
<b>P7</b>	La fase de codificación se extiende más de lo planificado.	1	3	0	1	0	0	0	3	0	0	8
<b>P8</b>	Los costos de los proyectos se desfasan.	0	0	0	0	0	0	0	0	1	0	1
<b>P9</b>	Lento engrane en el ingreso de personas nuevas en los proyectos.	0	2	0	1	1	1	2	2	0	1	10
<b>P10</b>	Las pruebas internas consumen un tiempo considerable en la codificación	3	2	0	3	3	0	2	2	0	0	15
	<b>y</b>	12	16	2	13	9	2	15	14	3	10	

**Fuente:** Elaboración propia

Establecida las relaciones entre los problemas; una forma de verificar la consistencia de la matriz se basa en calcular que no más del 30% de las celdas completadas se correspondan al valor 3. Es decir, que no puede existir más de un 30% de problemas con una correlación fuerte puesto que un alto porcentaje de los problemas se clasificarían como críticos y predominaría un solo cuadrante de resultados. En la figura No.23, para la presente investigación, se observa gráficamente la ponderación correspondiente a la valoración **3** de **21.74%**; por lo que se concluye que la elaboración de la matriz resultante del cuadro No.16, es consistente.



**Figura No.23.** Cantidad de valores asignados según ponderación

**Fuente:** Elaboración propia

#### 4. Construcción del Plano Cartesiano

Resuelta la matriz de influencia y dependencias, con la generación de los ejes X, Y; se debe calcular el promedio de la puntuación obtenida, cuadro No.17, que

representa el quiebre de los ejes respectivos, para determinar los cuatro cuadrantes y el peso ponderado de cada uno de los problemas analizados.

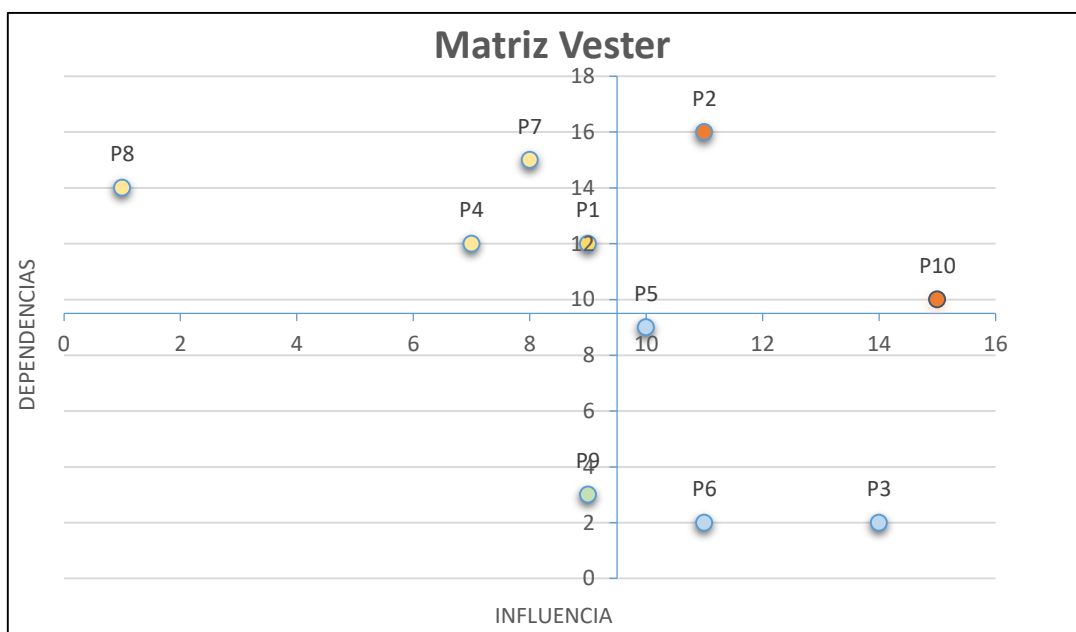
**Cuadro No.17.** Tabla de coordenadas y promedio dependencia e Influencia

	DESCRIPCION DEL PROBLEMA	X	Y
P1	Existen Errores en los Entregables	9	12
P2	Los tiempos de Entrega de los productos no siempre se respetan	11	16
P3	Surgen nuevos requerimientos durante la fase de codificación	14	2
P4	El tiempo de entrega limita las pruebas internas	7	13
P5	Existen cambios mayores debido a errores detectados en fases finales del proyecto	10	9
P6	Procesos no alineados a los protocolos definidos.	11	2
P7	La fase de codificación se extiende más de lo planificado.	8	15
P8	Los costos de los proyectos se desfasan.	1	14
P9	Lento engrane en el ingreso de personas nuevas en los proyectos.	10	3
P10	Las pruebas internas consumen un tiempo considerable en la codificación	15	10
<b>PROMEDIO DE INFLUENCIAS Y DEPENDENCIAS</b>		<b>9.60</b>	

**Fuente:** Elaboración propia

## 5. Clasificación de los Problemas

Con el promedio de influencias y dependencias, **9.6** para el trabajo de investigación, se traza una vertical sobre el eje X de influencia, y para el eje Y de dependencias. El resultado obtenido se muestra en el gráfico de la figura No.24.



**Figura No.24.** Matriz de Vester del Caso Dualbiz S.R.L.

**Fuente:** Elaboración propia

De la figura anterior, se divide el plano cartesiano en 4 cuadrantes, que se identifican con la clasificación de los problemas en: pasivos (1), críticos (2), indiferentes (3) y los activos (4); según se muestra en el cuadro resumen No.18.

**Cuadro No. 18.** Cuadrante resultado de Problemas, en Dualbiz S.R.L.

<b>Cuadrante 1</b> <b>Problemas Pasivos o</b> <b>Efectos de los problemas</b> <b>Críticos</b>	P1, P4, P7, P8
<b>Cuadrante 2</b> <b>Problemas Críticos o</b> <b>Centrales</b>	P2, P10
<b>Cuadrante 3</b> <b>Problemas Indiferentes</b>	P9
<b>Cuadrante 4</b> <b>Problemas Activos o</b> <b>Causas de los Problemas</b> <b>Críticos</b>	P3, P6, P5

**Fuente:** Elaboración propia



## 2.3 Conclusiones

Una vez finalizado el Diagnostico, mediante la Matriz de Vester, se determinan dos problemas críticos; **P2: “Los tiempo de entrega no siempre se respetan”** y **P10: “Las pruebas internas consumen un tiempo considerable en la codificación”**, candidatos para ser tomando en cuenta en la propuesta de solución al problema de investigación: Integración de una herramienta para la Generación Automática de Casos de Prueba en la fase de codificación de proyectos de desarrollo de *software* en Dualbiz S.R.L., para incidir positivamente en la curva de tiempo, a través de la detección temprana de errores en la fase mencionada. Así como también, con esta solución, se debe influenciar en problemas pasivos como **P1 “Existen errores en los entregables”** y **P4 “Los tiempos entrega limitan las pruebas internas”**.

## CAPÍTULO III – PROPUESTA

El presente capítulo inicia con la propuesta de solución al problema de investigación, explicando primeramente las razones tomadas en consideración en la selección de *IntelliTest* como herramienta para la Generación Automática de Casos de Prueba que se integrará a la solución, aplicada en el contexto de investigación, Dualbiz S.R.L. Posteriormente se detalla la estrategia para la integración y un plan de contingencia a la propuesta.

En la segunda parte del capítulo, bajo el esquema de investigación empírica propuesto por Wholin (Bocco, Cruz Lemus, & Piattini Velthuis, 2014), se desarrolla un pre-experimento para validar la propuesta y demostrar la hipótesis de investigación planteada.

### 3.1. Propuesta

#### 3.1.1. Por qué *Intellitest* como herramienta de integración

A partir de la sistematización realizada en el capítulo 1, sobre las herramientas para la Generación Automática de Casos de Prueba, se define utilizar *IntelliTest* como la herramienta a integrar dentro de la fase de codificación de proyectos de *software*, en Dualbiz S.R.L.; por sus ventajas, alcance y las especificaciones necesarias en el contexto donde se aplica.

- **Ventajas:** está incluida en la Herramienta de desarrollo *Visual Studio* (a partir de la versión 2015), manejo intuitivo, fácil integración al sistema de Gestión de Configuración que usa actualmente Dualbiz S.R.L. (*TFS-Team Foundation Server de Microsoft*)

Se puede ver fácilmente qué pruebas son las que fallan y agregar código para corregirlas. Se puede seleccionar las pruebas generadas que quiere guardar, en un proyecto de prueba, para proporcionar un conjunto de regresión. Cuando cambie el código, se puede volver a ejecutar *IntelliTest* para mantener sincronizadas las pruebas generadas con los cambios de código.

- **Alcance:** Funciona acoplándose a proyectos desarrollados en .NET solo para lenguaje c# y es acoplable al *TFS-Team Foundation Server* de *Microsoft*.
- **Especificaciones:**
  - Solo admiten código de C# que tenga como destino .NET Framework.
  - Son extensibles y admiten la emisión de pruebas en formato *MSTest*, *MSTest V2*, *NUnit* y *xUnit*.
  - No admiten la configuración x64.

### 3.1.2. Pasos para la integración

Se deben integrar primero a los proyectos desarrollados en .NET por ser éstos los que están en actual fase de codificación y tener un conjunto de requerimientos pendientes de ejecución.

Para realizar la integración de la herramienta, según se observa en la figura No. 25, se llevan a cabo las siguientes etapas:

1. **Integrar la Herramienta seleccionada *IntelliTest*** al sistema de Gestión de configuración actual *Team Foundation System* (TFS), por lo que se debe revisar los procedimientos necesarios para que los casos de pruebas generados

puedan proveer los resultados esperados, de la automatización, unido al repositorio de código.

Para realizar la integración se debieron realizar adecuaciones importantes sobre el proyecto, algunas de las cuales se detallan en los Anexos No.5 y No.6.



**Figura No.25.** Etapas de Integración de Herramienta Propuesta *IntelliTest*

**Fuente:** Elaboración propia

- 2. Documentar la integración de la herramienta *Intellitest*,** el uso y problemas frecuentes. para tenerla disponible al equipo de desarrollo en el Sistema de Gestión Documental de uso, *Sharepoint*.
- 3. Capacitar, al equipo de desarrollo, en el uso adecuado de la herramienta** para que lleven a cabo el proceso de integración de la herramienta a los proyectos de desarrollo de *software* y así obtener el resultado esperado.
- 4. Elaborar un plan de contingencia,** que permita garantizar continuidad en la ejecución de las tareas propuestas y retroalimentar el avance en el proceso de integración de la herramienta para la Generación Automática de Casos de

Prueba, al proceso de codificación de los proyectos de *software*, aun cuando se presente alguna situación ajena de carácter interno o externo a dicho proceso.

### 3.1.3. Plan de contingencia

Como se ha explicado anteriormente, el objetivo del plan de contingencia se centra en garantizar la continuidad del proceso de integración de la herramienta al proceso de codificación en los proyectos de desarrollo de *software*, en Dualbiz S.R.L. y retroalimentar el avance en la ejecución del mismo. Por ello, se plantean los siguientes aspectos para el seguimiento dentro la integración:

- Introducir casos con errores comunes, en la documentación de acceso general para el equipo de desarrollo.
- Verificar el seguimiento, dentro de objetivos pequeños relevados, mediante la hoja de trabajo que contiene aspectos importantes, cuadro No.19, que garantizan el control en la ejecución.

**Cuadro No.19.** Hoja de trabajo *IntelliTest*

Nro.	Pregunta
1	¿Qué requerimiento se avanzó?
2	¿Tuvo problemas en la ejecución de la herramienta?
3	Número de casos generados automáticamente
4	Clases cubiertas con la herramienta

**Fuente:** Elaboración propia

- Monitorizar el avance de los requerimientos con la herramienta de Gestión Integrada de la empresa (*TFS*) diariamente, para establecer un seguimiento interno al objetivo de la propuesta.

### 3.2. Procedimiento para el Experimento

Como destacan Judd y Kenny (1981), el propósito de toda investigación es conseguir una información exacta y generalizable, lo cual equivale a afirmar que sea válida. Cook y Campbell (1979) aportan una definición del término validez con la siguiente expresión: "Usaremos los conceptos de validez e invalidez para referirnos a la mejor aproximación disponible a la verdad o falsedad de las proposiciones" (p.37).

Para la ejecución del pre-experimento bajo el esquema propuesto:

$$\mathbf{G} \rightarrow \mathbf{O}_i \mathbf{X} \mathbf{O}_i$$

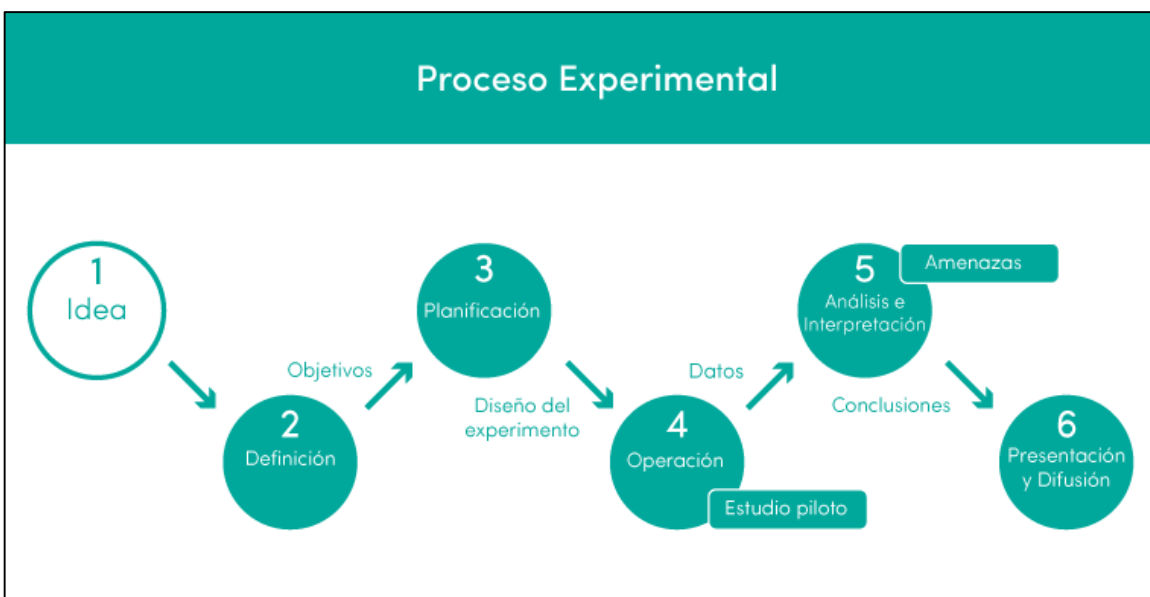
**G** → Grupo de Sujetos

**X** → Tratamiento, estímulo o condición experimental (presencia de algún nivel o modalidad de la variable independiente).

**O<sub>i</sub>** → Medición de los sujetos de un grupo (prueba, cuestionario, observación).

Si aparece antes del estímulo o tratamiento, se trata de una pre-prueba (previa al tratamiento). Si aparece después del estímulo, se trata de una pos-prueba (posterior al tratamiento).

En resumen, el esquema propuesto se puede definir como un solo grupo, que participa en la pre-prueba y pos-prueba, desde lo metodológico se define como caso único; a partir de un solo sujeto o unidad de observación. Se profundiza la articulación entre objetivos e instrumentos de recolección y análisis; su incidencia respecto de la muestra, considerándose las características de la misma para seguir los pasos que se describen a continuación, según la figura No.26.



**Figura No.26.** Proceso Experimental

**Fuente:** Cap. 3. Experimentos. (Genero Bocco, Cruz Lemus, & Piattini Velthuis, 2014)

- 1) **Definición del alcance:** En esta paso, se definen los objetivos del experimento: analizar, con el propósito de, con respecto a, desde el punto de vista de, en el contexto de.
- 2) **Planificación:** Al definir los objetivos del experimento, se realiza la planificación de éste para tener un conocimiento claro de la ejecución del mismo. La planificación se divide en 6 tareas:
  - I. **Selección del contexto:** En esta etapa, se determina el entorno en que se ejecuta el experimento.
  - II. **Formulación de hipótesis:** Consiste en expresar el objetivo del experimento a través de una hipótesis a probar. Se define una hipótesis de investigación ( $H_1$ ) y/o hipótesis nula ( $H_0$ ), para su validación se utilizan métodos de la estadística paramétrica.

- III. **Selección de variables:** A partir de la hipótesis definida en el paso anterior, se identifican: la variable independiente y las variables dependientes, de acuerdo al esquema propuesto del pre-experimento; se asume una definición conceptual para determinar la escala de medición y el rango de valores que pueden tomar.
  - IV. **Selección de sujetos:** Se realiza la selección del sujeto/grupo al cual se aplica los tratamientos del experimento.
  - V. **Elección del diseño:** Consiste en seleccionar el diseño apropiado que se utiliza como base para la realización del experimento.
  - VI. **Instrumentación:** Esta etapa tiene como objetivo describir los medios para realizar el experimento.
- 3) **Operación:** Una vez concluidos los pasos de diseño y planificación, se debe llevar a cabo la recolección de datos que se han de analizar posteriormente, para ello se ejecutan tres tareas:
- I. **Preparación:** Se debe comenzar el experimento con personas que deseen formar parte de él como sujeto de investigación.
  - II. **Ejecución:** Consiste en reunir a los sujetos que son parte del experimento para realizar la ejecución del experimento en un mismo lugar.
  - III. **Validación de datos:** se debe comprobar que los datos obtenidos en la ejecución son razonables y que se han recogido correctamente.
- 4) **Análisis e interpretación:** Una vez concluido el paso anterior, se realiza el análisis y la interpretación de los datos obtenidos del experimento.



- 5) **Presentación y difusión:** En este paso, se presenta los hallazgos del experimento.

### 3.3. Proceso Experimental

Para llevar a cabo el proceso experimental se ejecutan los pasos anteriormente descritos:

#### 3.3.1. Definición del alcance

Para dar cumplimiento al paso 1 del proceso experimental, para la presente investigación se determina utilizar el esquema  $G \rightarrow O_1 X O_2$  es decir, un solo grupo, pre-prueba y pos-prueba. En este diseño, un grupo es comparado consigo mismo en un contexto  $O_1$  y luego se introduce el cambio en un contexto  $O_2$ ; para establecer una línea base o punto referencial y ver la diferencia o impacto entre  $O_1$  y  $O_2$ .

$G \rightarrow$  Selección de desarrolladores del proyecto en fase de codificación.

$X \rightarrow$  Herramienta para la Generación Automática de Casos de Prueba:  
*IntelliTest*.

$O_1 \rightarrow$  Contexto Inicial: Enfoque Tradicional para realizar el desarrollo de los requerimientos.

$O_2 \rightarrow$  Contexto final: se introduce la Herramienta para la Generación Automática de Casos de Prueba, *Intellitest*.

#### 3.3.2. Planificación

En este paso de planificación se llevan a cabo las siguientes tareas:

## **I. Selección del Contexto**

Los objetos experimentales son una selección de parte del grupo de desarrolladores de Dualbiz S.R.L. y un conjunto de requerimientos que han sido seleccionados y ponderados como se detalla más adelante en la elaboración del pre-experimento.

Así como también, se muestra a continuación en el cuadro No 20, el conjunto inicial de requerimientos para el proyecto completo, donde se detalla Identificador del Desarrollador (ID. DESARR.), Identificador del Requerimiento (ID. REQ.), Descripción corta del requerimiento y Tiempo Estimado (T. Estimado). El proyecto tiene un tiempo estimado y fechas de entrega que ayudará a determinar desviaciones sobre las estimaciones.

**Cuadro No.20.** Requerimientos iniciales del proyecto

ID DESARR.	ID REQ.	Descripción corta del requerimiento	T. Estimado
D1	R1	Parametrización de feriados nacionales, regionales.	2
D1	R2	Adecuaciones para registro de empleado: aumentar tipo de sangre, licencia de conducir y fecha de vencimiento. Registrar referencias	2
D1	R3	Histórico de eventos para generación de planillas (reporte)	2
D1	R6	Realizar adecuaciones para registro de herederos, nombre, fecha de nacimiento, relación de dependencia, tipo de documento de identidad y número.	2
D1	R9	Programa para registro y seguimiento de los memorándums	3
D1	R15	Registro de impuestos por lote ABM	1
D1	R7	ABM para el Registro de Cargos (formato NAVI)	2
X	R8	Formulario Evaluación de Rendimiento (Formato NAVI)	3
D1	R16	Registro de desahucio y cálculo si existe	2
D2	R4	Realizar ABM para subir contratos bases	2
X	R11	Histórico de eventos para auditoria de aprobaciones (reporte)	2
D1	R12	Generador de archivo texto para pago de sueldos	2.5
D2	R13	Adecuaciones para registro de facturas y detalle de saldos crédito fiscal	2.5
X	R14	Impuestos asignados (reporte)	3
D2	R5	Realizar ABM para registro de póliza	3
D2	R10	Generación de la planilla impositiva, tomando en cuenta las UFVs	5
D1	R17	Carga RC-IVA desde Excel	4
D2	R18	Registro de primas extras por cumplimiento (ABM)	3
D2	R19	Registro de dimisión (ABM)	2
D2	R20	Registro de dependientes (adecuación sobre programa de registro empleado)	4
D1	R21	Parametrización de las UFV	2
D1	R22	Adecuaciones para el proceso de generación y aprobación y contabilización de la planilla (integrado a sistema propietario )	5
D2	R23	Generar planilla de salarios, con los campos que están en el modelo (NAVI)	5
D2	R24	Generación de liquidación y finiquito. Formato (NAVI)	5

**Fuente:** Elaboración propia

## II. Formulación de la Hipótesis de investigación

- **Hipótesis de investigación  $H_1$ :** Existe una mejora en la curva de tiempo a través de la detección temprana de errores en la fase de codificación de proyectos de desarrollo de software con la integración de herramientas para la Generación Automática de Casos de Prueba.
- **Hipótesis Nula  $H_0$ :** No existe una mejora en la curva de tiempo a través de la detección temprana de errores en la fase de codificación de proyectos de desarrollo de *software* con la integración de herramientas para la Generación Automática de Casos de Prueba.

## III. Selección de Variables

Se identifica como variable independiente los “Requerimientos Balanceados” (***RBN***); que son parte de los requerimientos en la fase de codificación, utilizando como criterio de ponderación, la complejidad para realizar el mismo. Estos requerimientos fueron balanceados siguiendo el criterio descrito en el siguiente cuadro No. 21.

**Cuadro No.21.** Criterio de complejidad para balancear requerimientos

Tipo de Requerimiento	Complejidad
Alta, baja y modificación	NIVEL 1
Reportes listados	NIVEL 1
Reportes cruzados o con cálculos	NIVEL 2
Formularios maestro/detalle	NIVEL 2
Transaccional	NIVEL 3

**Fuente:** Elaboración propia

Y como variable dependiente del experimento se identifica el “Tiempo de Ejecución” ( $T_{EJC}$ ) para cada uno de los Requerimientos Balanceados, utilizando las siguientes medidas:

**Tasa de Efectividad** ( $T_{EFEC}$ ) esta magnitud está relacionada al tiempo de ejecución del requerimiento versus tiempo estimado para el desarrollo del mismo. Con esto se quiere conocer una optimización en la misma o también revisar si los requerimientos están correctamente dimensionados.

Otra variable relacionada al experimento que no se realiza un cálculo directo en el proceso experimental es el porcentaje de código cubierto por las pruebas, en el anexo No. 4 se muestra un ejemplo de medición para el requerimiento 15.

#### **IV. Selección de Sujetos**

Actualmente en Dualbiz S.R.L. los equipos de desarrollo se dividen por área, como se explicó anteriormente, dependiendo de la tecnología y del proyecto. Para desarrollar esta prueba se decide trabajar con un proyecto en actual desarrollo que consta de 3 personas encargadas de realizar el conjunto de cambios para el proyecto en cuestión, los mismos tuvieron que ser capacitados en el uso y también en la lógica de trabajo de la herramienta, propuesta para la investigación.

#### **V. Elección del Diseño**

Con los requerimientos descritos en la tarea de selección del contexto, cuadro No.20, se define primeramente catalogarlos según los niveles detallados en el cuadro No. 21. Para balancear los requerimientos se toma en cuenta el nivel de complejidad

y la suma total del tiempo estimado de ambos conjuntos, con el fin de realizar la ponderación de Tiempo Estimado de Ejecución y Nivel de Complejidad.

De esta manera, se obtiene un conjunto de requerimiento balanceados  $C_{R1}$ , según se muestra en el cuadro No.22, para el contexto  $O_1$ : “Manera Tradicional para ejecutar los requerimientos”.

**Cuadro No.22.** Requerimientos balanceados contexto  $O_1$

Nivel	Requerimiento	T. Estimado	Tiempo Ejec
<b>NIVEL 1</b>	R1	1.0	0.80
	R6	2.0	1.50
	R7	2.0	1.75
	R15	1.0	1.35
<b>NIVEL 2</b>	R2	2.5	2.30
	R3	2.5	2.55
	R4	2.5	3.20
	R5	3.0	3.06
<b>NIVEL 3</b>	R8	4.0	3.56
	R9	4.0	3.90
	R10	4.5	5.25
	R24	5.0	6.75
<b>Total</b>		35.0	<b>35.97</b>

**Fuente:** Elaboración propia

De la misma manera, se trabaja sobre los requerimientos restantes para el contexto  $O_2$ ; obteniéndose los requerimientos balanceados  $C_{R2}$ , mostrados en el cuadro No.23. Los requerimientos, en este contexto, se ejecutan con la ayuda de la Herramienta *InteliTest* para la Generación Automática de Casos de Prueba, seleccionada de acuerdo a los diferentes criterios detallados en la propuesta.

**Cuadro No.23.** Requerimientos balanceados contexto O<sub>2</sub>

Nivel	Requerimiento	T.Estimado	Tiempo Ejec
<b>Nivel 1</b>	R11	1.0	0.78
	R16	1.5	1.56
	R19	2.0	1.35
	R21	2.0	1.06
<b>Nivel 2</b>	R12	2.5	1.55
	R13	2.5	1.70
	R14	2.5	1.75
	R18	3.0	2.58
<b>Nivel 3</b>	R17	4.0	3.65
	R20	4.0	3.80
	R22	5.0	6.75
	R23	5.0	5.55
<b>Total</b>		35.0	<b>32.03</b>

**Fuente:** Elaboración propia

## VI. Instrumentación

Como última tarea de la planificación, para la instrumentación se determina utilizar *t* de *Student* como método de comprobación de hipótesis. Utilizando la información obtenida a partir del Sistema de Gestión de Configuración (*TFS*) para construir la tabla de muestras; con el tiempo de ejecución de ambos *Sprint*, es decir *Sprint 1* y *Sprint 2* para el contexto **O<sub>1</sub>** y el contexto **O<sub>2</sub>**, respectivamente; con cada uno de los requerimientos balanceados en el contexto correspondiente.

Una vez recolectada y estructurada la información, se utiliza el método *t* de *Student* para la confirmación de la hipótesis de trabajo, propuesta para el pre-experimento.

*t* de *Student* es una distribución de variabilidad que surge del problema de estimar la media de la población normalmente distribuida, cuando el tamaño de la muestra es pequeña. Aparece de manera natural, realizar la prueba *t* de *Student*

para la determinación de las diferencias entre dos varianzas muestrales y para la construcción del intervalo de confianza, diferencia entre las parte de dos poblaciones cuando se desconoce la desviación típica de una población y ésta debe ser estimada a partir de los datos de la muestra (Soler & Perez, 2008).

Para la ejecución del procedimiento de análisis con el método *t* de *Student*, se necesita seguir los siguientes pasos:

**Paso 1. Planteamiento de Hipótesis.**

**Paso 2. Modelo Experimental**, de acuerdo al tipo de estudio de la investigación: longitudinal, para muestras menores a 30. Se selecciona Prueba *t* para muestras relacionadas con coeficiente de Wilcoxon.

**Paso 3. Nivel Alfa**, o también denominado nivel de significancia o margen de error para investigaciones, debe ser siempre menor o igual a 0.05 o 5%, es decir tratar de determinar un valor de comprobación con aceptación de hasta 95%.

**Paso 4. Prueba de Normalidad** o Grado de normalidad, es decir, si ambas muestras tienen un comportamiento normal y están dentro de un rango de significancia. Existen dos métodos para muestras menores a 30 Shapiro Wilk y para mayores a 30 Kolmogorov – Smirnov.

**Paso 5. Decisión Estadística**, comparando el cálculo de P-valor para el modelo de estudio seleccionado, se debe realizar la comprobación estadística de la prueba realizada.

Alguno de los valores necesarios para la revisión y análisis en este proceso experimental, se detalla a continuación:



**Media de Distribución (M)**, es valor de tendencia central o media aritmética del conjunto de valores obtenidos de la experimentación o prueba.

**Desviación Estándar de la Muestra (V)**, es la desviación con respecto a la media central obtenida de la muestra.

**P-Valor**, para la validación estadística de la curva normal.

### 3.3.3. Operación

- I. **Preparación**, como se menciona en pasos anteriores puntualmente en la selección de sujetos; una vez integrado los Casos de Prueba Generados por *IntelliTest* al Sistema de Gestión de Configuración y revisado todos los detalles para que la herramienta pueda funcionar de manera autónoma, se ve conveniente que el grupo de desarrollo, inicie el *Sprint* 1 sin conocimiento de la herramienta, para que luego el mismo grupo entre en la fase de capacitación y conozca el uso adecuado y detalles de la herramienta seleccionada, *Intellitest*. Esto, para no inducir o modificar el ambiente del contexto **O<sub>1</sub>** en la ejecución de las tareas del modo tradicional y tener un nuevo contexto **O<sub>2</sub>** de la prueba con el uso de la herramienta.
- II. **Ejecución**, al contar con un mismo grupo, el presente estudio no se lleva a cabo de manera transversal, sino longitudinal; con los datos obtenidos de la Instrumentación se procede a la comprobación de hipótesis por el método t de *Student*, siguiendo los pasos descritos a continuación:

**Paso 1. Hipótesis** (se utiliza la hipótesis central de la investigación):

- **Hipótesis de investigación  $H_1$ :** Existe una mejora en la curva de tiempo a través de la detección temprana de errores en la fase de codificación de proyectos de desarrollo de software con herramientas para la Generación Automática de Casos de Prueba.
- **Hipótesis Nula  $H_0$ :** No existe una mejora en la curva de tiempo a través de la detección temprana de errores en la fase de codificación de proyectos de desarrollo de *software* con herramientas para la Generación Automática de Casos de Prueba.

**Paso 2. Modelo Experimental**, para la prueba se determina utilizar un modelo longitudinal de un mismo grupo con variable numérica para muestras relacionadas y medidas distintas.

**Paso 3. Nivel Alfa;** Se elige 0.05 como nivel de significancia equivalente a 5% de error, esto es un estándar para investigaciones científicas.

**Paso 4. Supuesto de Normalidad.** Para la prueba de normalidad se debe calcular la media y desviación estándar para cada una de las muestras obtenidas en cada contexto, obteniendo los siguientes resultados:

### **Contexto $O_1$ (Sprint 1)**

Se obtienen los valores de la media y desviación estándar, del presente contexto, para realizar los cálculos de la Distribución Normal del contexto  $O_1$ , mostrados los resultados en el cuadro No.24.

$$M1 = \frac{\sum(T.Ejecucion Req. Contexto O1)}{Total Requerimientos} = 2.9975$$

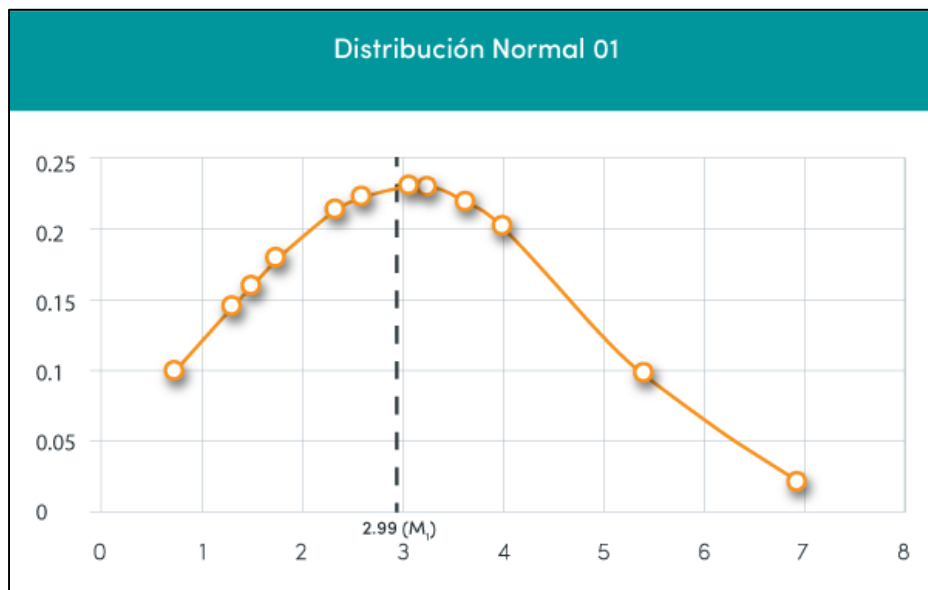
$$d1 = \sqrt{\frac{\sum_{i=1}^N (TEJEC.REQ_i - M1)^2}{Total\ Requerimientos - 1}} = 1.7151$$

**Cuadro No.24.** Distribución Normal contexto O<sub>1</sub>

Requerimiento	T.Ejec	Distribución Normal
R1	0.80	0.102365233
R6	1.35	0.146639966
R7	1.50	0.158881924
R15	1.75	0.178537687
R2	2.30	0.214137782
R3	2.55	0.224813709
R4	3.06	0.232442924
R5	3.20	0.230981820
R8	3.56	0.220419088
R9	3.90	0.202526586
R10	5.25	0.098194305
R24	6.75	0.021242194

**Fuente:** Elaboración propia

Con los resultados del cuadro anterior, se obtiene la gráfica, de la figura No.26, de Distribución Normal del contexto O<sub>1</sub>.



**Figura No.27.** Distribución Normal de la muestra contexto O<sub>1</sub>

**Fuente:** Elaboración propia.

### Contexto O<sub>2</sub> (Sprint 2)

Se realizan los cálculos de la media y desviación estándar del presente contexto, para obtener los valores de la Distribución Normal del contexto O<sub>2</sub> que se muestran en el cuadro No.25.

$$M2 = \frac{\sum(T.Ejecucion\ Req.\ Contexto\ O2)}{Total\ Requerimientos} = 2.6691$$

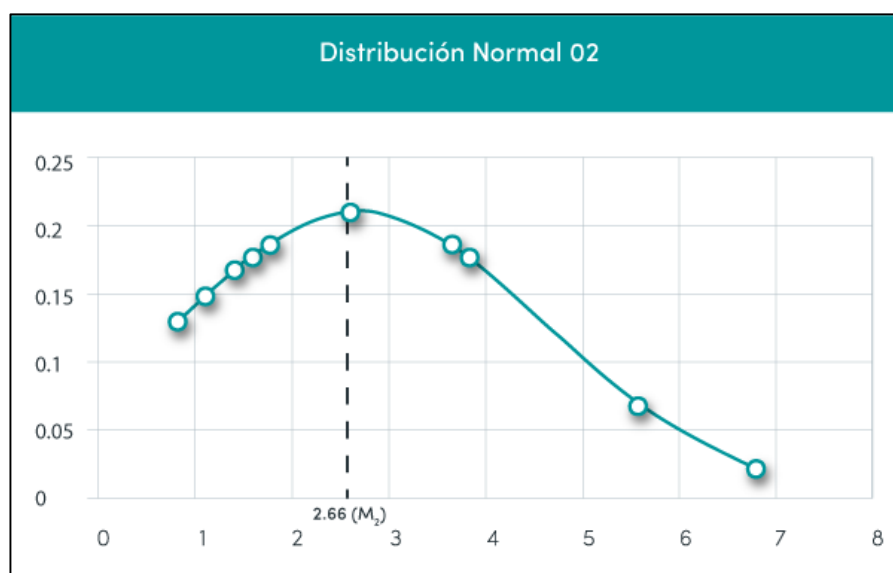
$$d2 = \sqrt{\frac{\sum_{i=1}^N (TEJEC.REQi - M2)^2}{Total\ Requerimientos - 1}} = 1.8940$$

**Cuadro No. 25.** Distribución Normal contexto O<sub>2</sub>

Requerimiento	T.Ejec	Distribución Normal
R11	0.78	0.128082465
R16	1.06	0.146818850
R19	1.35	0.165266853
R21	1.55	0.176890583
R12	1.56	0.177440824
R13	1.70	0.184784718
R14	1.70	0.184784718
R18	2.58	0.210397511
R17	3.65	0.184199722
R20	3.80	0.176244582
R22	5.55	0.066246491
R23	6.75	0.020676967

**Fuente:** Elaboración propia

Con los resultados del cuadro anterior, se obtiene la gráfica, de la figura No.28, de Distribución Normal del contexto O<sub>2</sub>.



**Figura No.28.** Distribución Normal de la muestra contexto O<sub>2</sub>

**Fuente:** Elaboración propia.

### Paso 5. Decisión Estadística

Con los resultados obtenidos del paso anterior, se calcula el P-valor, utilizando la herramienta *Excel*, “determinación de la prueba t de *Student* para muestras emparejadas”, obteniéndose los resultados que se muestran en el cuadro No.26.

**Cuadro No.26.** Resultados *Excel*. Prueba t de *Student* para muestras emparejadas

	Variable 1	Variable 2
<b>Media</b>	<b>2.9975</b>	<b>2.6691</b>
<b>Varianza</b>	<b>2.9417</b>	<b>3.5873</b>
Observaciones	12	12
Coeficiente de correlación de Pearson	0.97042602	
Diferencia hipotética de las medias	0	
Grados de libertad	11	
Estadístico t	2.4023903	
P(T<=t) una cola	0.01754193	
Valor crítico de t (una cola)	1.79588482	
<b>P(T&lt;=t) dos colas</b>	<b>0.02508385</b>	
Valor crítico de t (dos colas)	2.20098516	

**Fuente:** Elaboración propia

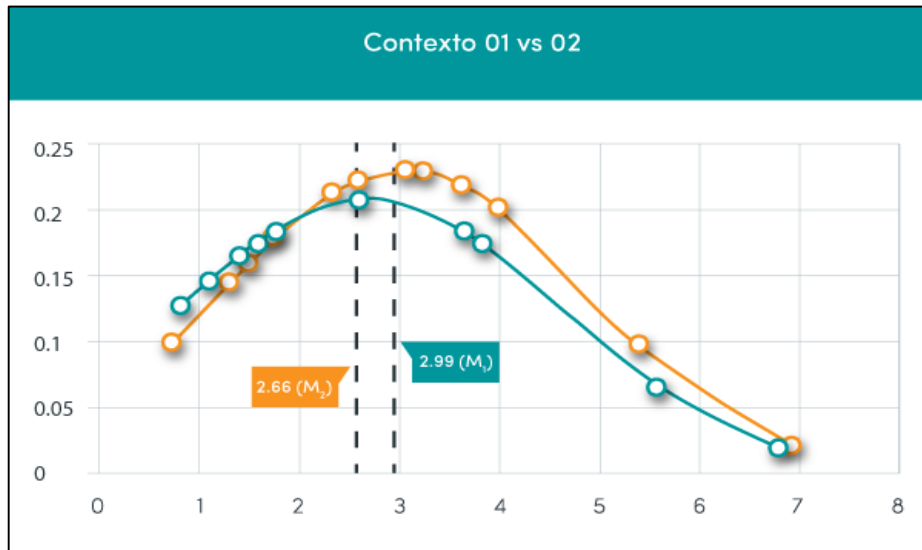
P-valor  $\leq$  Nivel Alfa, se rechaza la hipótesis nula  $H_0$  y se acepta la hipótesis de investigación  $H_1$ .

$0.01715 \leq 0.05$ , por consiguiente se acepta  $H_1$ : “Existe una mejora en la curva de tiempo a través de la detección temprana de errores en la fase de codificación de proyectos de desarrollo de *software* con la integración de herramientas para la Generación Automática de Casos de Prueba”.

### 3.2.4. Análisis e Interpretación

De acuerdo con los resultados obtenidos por la herramienta *Excel* para el análisis de la muestra con la distribución t de *Student* de ambos contextos  $O_1$  y  $O_2$ , hay una diferencia en las medias de los tiempos de ejecución de los requerimientos del *Sprint 1* y *Sprint 2*, con un P-valor menor al valor de significancia; por lo tanto se concluye que: “Existe una mejora en la curva de tiempo a través de la detección temprana de errores en la fase de codificación de proyectos de desarrollo de *software* con la integración de herramientas para la Generación Automática de Casos de Prueba, en Dualbiz S.R.L”.

De hecho, como se observa en la figura No.29, la **Media del Tiempo de Ejecución** disminuye de 2.99 a 2.62 días, con una desviación estándar menor.



**Figura No.29.** Distribución Normal de la muestra contexto O2

**Fuente:** Elaboración propia.

## CONCLUSIONES

- 1 La sistematización realizada a diferentes autores permitió elaborar la fundamentación teórica del objeto de estudio: Herramientas para la Generación Automática de Casos de Prueba, con un enfoque histórico-lógico se transitó desde el concepto básico de las pruebas y se profundizó en cada uno de las ellas, definiendo *IntelliTest* como la herramienta a utilizar en la propuesta por sus ventajas, alcance y las especificaciones necesarias en el contexto donde se aplica.
- 2 Con la aplicación de los instrumentos de investigación, entrevista y encuesta se diagnosticó la situación actual de la fase de codificación en Dualbiz S.R.L., elaborada la lista de problemas candidatos, se utilizó la Matriz de Vester para verificar la motricidad de los problemas y de esta manera se obtuvieron los problemas críticos a resolver en la propuesta: Los tiempos de entrega no siempre se respetan y las pruebas internas consumen un tiempo considerable en la codificación.
- 3 Se integró *Intellitest* al Sistema de Gestión de Configuración en Dualbiz S.R.L. como la herramienta seleccionada para la Generación Automática de Casos de Prueba en la fase de codificación; para ello se ejecutaron 4 etapas: integración, documentación, capacitación y elaboración de la hoja de trabajo *IntelliTest*.
- 4 Se validó la mejora en la curva de tiempo para la ejecución de los requerimientos en fase de codificación a través del uso una Herramienta para la Generación Automática de Casos de Prueba en Dualbiz S.R.L. utilizando el método t de *Student* propio de la estadística paramétrica.



## RECOMENDACIONES

- 1 Se recomienda realizar un análisis de la selección del contexto, es decir, los proyectos candidatos y los sujetos que intervienen, de esta manera minimizar los problemas en la ejecución del proceso de integración de la herramienta.
- 2 Es recomendable tener en cuenta una lista factores internos y externos así como también un plan de contingencia y guía de trabajo diaria para la recolección de la información.
- 3 Para mejorar el análisis de efectividad se sugiere ampliar la muestra experimental o ampliar la investigación elevándola a un proceso experimental con grupo de control.
- 4 Se sugiere utilizar otras herramientas para la Generación Automática de Casos de Pruebas para futuros trabajos de investigación tal como *EvoSuite* que permite ser utilizada en otros entornos para trabajar con *JAVA*.
- 5 Socializar los resultados de la investigación, en la comunidad científica y estudiantil de la Unidad de Postgrado y la Facultad de Ciencias de la Computación y Telecomunicación; como base para futuros trabajos sobre herramientas de automatización de procesos de prueba en otros niveles del ciclo de vida del *software*.

## REFERENCIA BIBLIOGRÁFICA

- Allen Goldberg, T. W. (1994). *ACM Asociation for Computing Machinery*. Obtenido de <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.6306&rep=rep1&type=pdf>
- Beizer, B. (1990). *Black box testing, white box testing*. En B. Beizer, *Software testing Techniques*.
- Bertolino, A. (2007). *Software testing research: Achievements, challenges, dreams*. En A. Bertolino, *Future of Software Engineering*.
- Blanco, R.; Diaz, E. & Tuya, J. (2006). *Generación automática de casos de prueba mediante*. Revista Española de Innovación, Calidad e Ingeniería del Software.
- Bocco, M. G., Cruz Lemus, J. A. & Piattini Velthuis, M. (2014). *Métodos de investigación del Software*. Madrid: Ra-Ma.
- Crosby, P. B. (2012, 08 15). *Ejemplode.com*. Obtenido de Cero Defectos: [http://www.ejemplode.com/58-administracion/3968-ejemplo\\_de\\_cero\\_defectos.html](http://www.ejemplode.com/58-administracion/3968-ejemplo_de_cero_defectos.html)
- Falgueras, B. C. (2003). *Ingeniería del Software*. Catalayuna - Universidad de Oberta: UOC.
- Fraser, G., & Arcuri, A. (2013). *Whole Test Suite Generation*. IEEE. Obtenido de <https://dl.acm.org/citation.cfm?id=2478706>
- Genero Bocco, M., Cruz Lemus, J. & Piattini Velthuis, M. (2014). *Métodos de Investigación en Ingeniería del Software*. Ra-Ma.

- Glenford, J. M. (1979). *The Art of Software Testing*. John Wiley & Sons, Inc.
- Glover, F., & Laguna, M. (1977). *Fundamental of Scatter Search and Path Relinking*.
- Gorden Fraser, A. A. (2011). *Whole Test Suite Generation*. Proceedings of the 19th ACM SIGSOFT symposium and the 13th European Conference on Foundations of software engineering.
- Helperin, D., & Hetzel, B. (1998). *The Growth of Software Testing*. Communications of the ACM, Volume 31 PAgos 687-695. Obtenido de ACM IEE.
- Hernández Sampieri, R. (2012). *Metodología de la Investigación*. Mc Graw Hill.
- José Campos, A. A. (2014). *Continuous Test Generation: Enhancing Continuous*. University of Sheffield, Departament of Computer Science, UK. Obtenido de <http://www.ev-suite.org>: [http://www.ev-suite.org/wp-content/papercite-data/pdf/ase14\\_ctg.pdf](http://www.ev-suite.org/wp-content/papercite-data/pdf/ase14_ctg.pdf)
- Lisa Crispin, J. G. (2009). *Agile Testing* (2009 ed.). Mike cohn, Signature Series.
- Luo, L. (1990). *Software Testing Techniques, Technology Maturation and Research Strategy*. Obtenido de <https://www.cs.cmu.edu/~luluo/Courses/17939Report.pdf>
- Miller, E. F. (2012). *Introduction to Software Testing Technology*.
- Northrop, R. B. (2011). *Introduction to Complexity and Complex Systems*. CRC press.
- Osherov, R. (2009). *The Art Of Unit Testing*. Manning Publications.
- Pressman, R. (2005). *A practitioner's approach*. En *Software engeeniering*.
- Satpathy, T., & Sutherland, J. (2016). *Una guía para el SCRUMstudy*.

Sebastian Bauersfeld, S. W. (2011).

*[https://www.researchgate.net/publication/221469955\\_A\\_Metaheuristic\\_Approach\\_to\\_Test\\_Sequence\\_Generation\\_for\\_Applications\\_with\\_a\\_GUI](https://www.researchgate.net/publication/221469955_A_Metaheuristic_Approach_to_Test_Sequence_Generation_for_Applications_with_a_GUI)*. Obtenido de Research Gate.

Soler, C., & Perez, J. (2008). *Nociones de estadística aplicadas a la investigación*.

UCP.

Sommerville, I. (2012). *Ingeniería del Software*. Addison Wesley.

Warren, G., & Saisang, C. (2015). *Generar pruebas unitarias para el código con*

*IntelliTest*. Obtenido de <https://docs.microsoft.com/es-es/visualstudio/test/generate-unit-tests-for-your-code-with-intellitest>

## BIBLIOGRAFÍA

1. Gerardo Miranda Quintana (2014). *Estudio de caso sobre Herramienta de Generación Automática de Casos de Prueba*. CIBSE2014 Conference of Software Engineering Universidad ORT Uruguay.
2. Pranali Mahadic, Seongbuk Gu (2016). *A review for Automated Test Case Generation*. International Journal of Software Engineering and its Applications Vol 10.
3. Praveen Ranjan, Rahul Khandelwal (2012). *Automated Test Data Generation using Cukoo Search and Tabu Search*. IEE Publication Proceeding of World Congrees on Nature & Biologically Inspired Computing.(jisisys-2012-0009)
4. Raúl Gustavo Eid Ayala Ph.D. (2007). *Estructura de una Tesis*. Universidad Javierana.

## ANEXOS

### **Anexo No.1.** Entrevista a Gerentes de Desarrollo y Operaciones

Estimado colega,

La presente entrevista, forma parte de los instrumentos de investigación que se aplican en el proceso de diagnóstico para un trabajo de investigación de tesis de maestría. Tiene por objetivo obtener información sobre las causas o problemas asociados a la fase de codificación y las incidencias en los productos de comercialización, en Dualbiz S.R.L.

Agradecemos de antemano su colaboración.

1. ¿Cuántos Equipos de Desarrollo tienen actualmente?
2. ¿Qué cantidad de personas integran estos equipos?
3. ¿Qué plataforma de desarrollo se utiliza para los proyectos?
4. Seleccionar los tipo de pruebas que se realizan actualmente sobre los proyectos?
  - a) Pruebas de Caja Blanca
  - b) Pruebas de Caja Negra
  - c) Pruebas Funcionales
  - d) Pruebas Exploratorias
5. ¿En caso de existir pruebas de caja blanca. Qué tipo de herramientas se utilizan para llevar a cabo las mismas?
6. ¿Existen herramientas de Gestion de Integración?
7. ¿Existen bitácoras de proyectos asociados a los mismos?
8. ¿Tiene conocimiento de defectos en proyectos realizados?

9. ¿Qué problemas puede detallar para los desfase de los proyectos?
  - a. ¿Qué planes futuros se tienen definidos sobre los problemas detallados?
5. Se manejan herramientas de gestión para la documentación de actividades o de los problemas detectados?

## Anexo No.2. Encuesta a desarrolladores

Estimado colega,

La presente encuesta, forma parte de los instrumentos de investigación que se aplican en el proceso de diagnóstico para un trabajo de investigación de tesis de maestría. Tiene por objetivo obtener información sobre las causas o problemas asociados a la fase de codificación y las incidencias en los productos de comercialización, en Dualbiz S.R.L. Así como, conocer aspectos relativos a la Generación Automática de Casos de Pruebas en proyectos de desarrollo de *software*.

Agradecemos de antemano su colaboración.

1. ¿Cuál es el cargo actual que desempeña? \_\_\_\_\_
2. ¿Cuáles son las actividades principales que realiza?  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
3. ¿En qué tipos de proyecto trabaja usualmente?  
\_\_\_\_\_  
\_\_\_\_\_
4. ¿Cuál es el entorno de trabajo (IDE) para codificar que utiliza?  
\_\_\_\_\_
5. ¿Tiene algún problema en el desempeño de las funciones descritas?  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
6. ¿Tiene conocimientos sobre herramientas para el desarrollo de Casos de Pruebas?  
Si \_\_\_\_\_ No \_\_\_\_\_  
a. ¿Qué nivel de uso usted cree que tiene para utilizar estas herramientas?  
ALTO\_\_\_\_\_ INTERMEDIO\_\_\_\_\_ BAJO\_\_\_\_\_  
b. ¿Qué herramienta(s) utiliza para la Generación de Casos de Prueba?
7. Está familiarizado con la terminología Recubrimiento de Código.



Si \_\_\_\_\_ No \_\_\_\_\_

8. Está familiarizado con el término Nivel de Detección de Errores

Si \_\_\_\_\_ No \_\_\_\_\_

9. Los proyectos actuales tienen un grado de Recubrimiento de Código (RC), determine en qué porcentaje?

0 \_\_\_\_\_  $0 < RC < 50\%$  \_\_\_\_\_  $RC > 75\%$  \_\_\_\_\_

10. ¿Cree que el tiempo para el desarrollo de pruebas internas para la codificación es dimensionado correctamente?

\_\_\_\_\_

11. Mencione los problemas que usted cree, influyen para generar retrasos en la entrega de proyectos.

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_






\_\_\_\_\_

\_\_\_\_\_

### Anexo No.3. Operacionalización del Objeto de Estudio

VARIABLE	DEFINICION OPERACIONAL	DIMESION	INDICADOR
<b>1. CODIFICACION</b>	La codificación es la fase del desarrollo de <i>Software</i> que involucra un equipo de trabajo, para traducir los requerimientos levantados en fases anteriores, a código que pueda ser utilizado por la máquina.	<b>1.1 Código</b>	Líneas de código, que se encuentran en un algoritmo
		<b>1.2 Tiempo de Ejecución</b>	Tiempo empleado para la implementación de un requerimiento de los proyectos en fase de codificación.
		<b>1.3 Equipo</b>	Personas en el equipo
<b>2. CASOS DE PRUEBA (Pruebas Unitarias)</b>	Conjunto de condiciones o variables bajo las cuáles un analista determinará si una aplicación, un sistema <i>software</i> ( <i>software system</i> ), o una característica de éstos es parcial o completamente satisfactoria. Ayudan a dar un porcentaje de recubrimiento de código y también brinda estadísticas para detección de errores	<b>2.1 Recubrimiento de código</b>	Código cubierto por Casos de Pruebas
		<b>2.2 Habilidad de Detección de Errores</b>	Nivel de habilidad de Detección de Errores, criterio utilizado por el tipo de Generación de Pruebas Automatizadas

**Anexo No.4.** Ejemplo de análisis de cobertura de código para la Clase Dimisión (BrhOrgDim)

Code Coverage Results					
Usuario_LAPTOPALC 2018-03-31 12_20_10.ci     					
Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Bloc...	Covered (% Bloc...	
▷ brhOrgCnt	146	91.25%	14	8.75%	
▷ brhOrgCom	184	100.00%	0	0.00%	
▷ brhOrgCom.<>c	11	100.00%	0	0.00%	
▲ brhOrgDim	87	61.70%	54	38.30%	
⊗ PenditesXLiq...	4	33.33%	8	66.67%	
⊗ agregar(RRH...	30	85.71%	5	14.29%	
⊗ brhOrgDim()	0	0.00%	4	100.00%	
⊗ buscar(long)	6	42.86%	8	57.14%	
⊗ buscarDimXP...	4	33.33%	8	66.67%	
⊗ buscarXContr...	4	33.33%	8	66.67%	
⊗ eliminar(long)	23	82.14%	5	17.86%	
<a href="#">Code Coverage Results</a> <a href="#">Output</a> <a href="#">Error List</a> <a href="#">Web Publish Activity</a>					

**Anexo No.5.** Integración *IntelliTest* sobre el Proyecto de Barrido

Configuration Manager?×

Active solution configuration: TESTING

Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
RRHH.BAS.BR	TESTING	Any CPU	<input checked="" type="checkbox"/>	<input type="checkbox"/>
RRHH.BAS.BR.Tests	Debug	Any CPU	<input checked="" type="checkbox"/>	<input type="checkbox"/>
RRHH.BAS.DA	TESTING	Any CPU	<input checked="" type="checkbox"/>	<input type="checkbox"/>

**Anexo No.6.** Cambios para compilación directa desde la herramienta y acceso a Base de Datos

```
Ant.cs ExcelUtility.cs BrExceptionHandler.cs xbCnxMetodo.cs brhOrgDimTest.cs brhOrgDimTest.modificarTest.g.cs brhOrgDimTest.agregarTest.g.cs _cFactory.cs
HH.BAS.DA RRHH.BAS.DA.Factory GetRRHHEntities

21
22
23
24
25
26
27 99+ references | Administrador, 59 days ago | 1 author, 2 changes | 2 incoming changes | 3 work items
28 public static RRHHEntities GetRRHHEntities
29 {
30     get {
31         #if (TESTING)
32             if (_ExcpMgr == null) _ExcpMgr = CnxEntity.EntLibExceptionHandler.getInstancia();
33             if (CnxEntity.xCnxMetodo.stringConexion == null) CnxEntity.xCnxMetodo.ArmarmCadenaConexion("bd_teCorp");
34         #endif
35
36         lock (_EntityLocker)
37         {
38             return new RRHHEntities(CnxEntity.xCnxMetodo.stringConexion);
39         }
40     }
41 }
42
43
```