



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Dirección General de Estudios de Posgrado
Facultad de Ingeniería de Sistemas e Informática
Unidad de Posgrado

**Método para reducir pruebas de regresión de software
basado en acceso a los datos en entornos de desarrollo
iterativo incremental**

TESIS

Para optar el Grado Académico de Doctor en Ingeniería de
Sistemas e Informática

AUTOR

Mg. Raúl Hernán ROSERO MIRANDA

ASESOR

Dr. Glen Darío RODRÍGUEZ RAFAEL

Lima, Perú

2021



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Rosero, R. (2021). *Método para reducir pruebas de regresión de software basado en acceso a los datos en entornos de desarrollo iterativo incremental*. [Tesis de doctorado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Unidad de Posgrado]. Repositorio institucional Cybertesis UNMSM.

METADATOS COMPLEMENTARIOS	
Datos del Autor	
Nombres y Apellidos	Raúl Hernán Rosero Miranda
DNI Carné de extranjería emitido en Perú Pasaporte Cédula de identidad	1707517551
URL de ORCID	https://orcid.org/0000-0002-2315-9773
Datos del Asesor	
Nombres y Apellidos	Glen Darío Rodríguez Rafael
DNI Carné de extranjería emitido en Perú Pasaporte Cédula de identidad	10062042
URL de ORCID	https://orcid.org/0000-0002-4471-3198
Datos de investigación	
Línea de Investigación	Ingeniería de sistemas e informática
Grupo de investigación	Inteligencia Artificial
Agencia de financiamiento	Financiamiento propio
Ubicación Geográfica de la investigación	Escuela Superior Politécnica de Chimborazo - ESPOCH Ecuador – Chimborazo – Riobamba Coordenadas: 1° 39' 28.57" S, 78° 40' 34.83" W
Año o rango de años en que se realizó la investigación	2013 - 2021
URL de la disciplina: OCDE	2.02.04 - Ingeniería de sistemas y comunicaciones URIS: https://purl.org/pe-epo/ocde/ford#2.02.04



Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática
Vicedecanato de Investigación y Posgrado
Unidad de Posgrado

ACTA DE SUSTENTACIÓN VIRTUAL DE TESIS PARA OPTAR EL GRADO ACADÉMICO DE DOCTOR EN INGENIERÍA DE SISTEMAS E INFORMÁTICA

A los siete (07) días del mes de marzo del 2021, siendo las dieciséis y treinta horas, se reunieron en la sala virtual meet.google.com/fjp-mcdr-oqx, el Jurado de Tesis conformado por los siguientes docentes:

Dr. Frank Edmundo Escobedo Bailón (Presidente)
Dra. Luz Sussy Bayona Oré (Miembro)
Dr. Javier Arturo Gamboa Cruzado (Miembro)
Dr. Glen Darío Rodríguez Rafael (Miembro Asesor)

Se inició la Sustentación invitando al candidato a Doctor **Raúl Hernán Rosero Miranda**, para que realice la exposición oral y virtual de la tesis para optar el Grado Académico de Doctor en Ingeniería de Sistemas e Informática, siendo la Tesis intitulada:

"Método para Reducir Pruebas de Regresión de Software Basado en Acceso a los Datos en Entornos de Desarrollo Iterativo Incremental"

Concluida la exposición, los miembros del Jurado de Tesis procedieron a formular sus preguntas que fueron absueltas por el graduando; acto seguido se procedió a la evaluación correspondiente, habiendo obtenido la siguiente calificación:

17 (DIECISIETE) muy bueno

Por tanto el Presidente del Jurado, de acuerdo al Reglamento General de Estudios de Posgrado, otorga al Magíster **Raúl Hernán Rosero Miranda** el Grado Académico de Doctor en Ingeniería de Sistemas e Informática

Siendo las 18:10 horas, el Presidente del Jurado de Tesis da por concluido el acto académico de Sustentación de Tesis.

Dr. Frank Edmundo Escobedo Bailón
(Presidente)

Dra. Luz Sussy Bayona Oré
(Miembro)

Dr. Javier Arturo Gamboa Cruzado
(Miembro)

Dr. Glen Darío Rodríguez Rafael
(Asesor)

1. Código ORCID del Autor: 0000-0002-2315-9773

2. Código ORCID del Asesor: 0000-0002-4471-3198

3. Grupo de Investigación: Inteligencia Artificial

4. Institución que Financia la Investigación:

Ninguna

5. Ubicación Geográfica:

Escuela Superior Politécnica de Chimborazo ESPOCH

Ecuador – Chimborazo - Riobamba,

Coordenas: 1° 39' 28.57" S, 78° 40' 34.83" W

6. Año o rango de Años que abarco la investigación:

2013 - 2021

Cédula Ecuatoriana N°. 170751755-1

DEDICATORIA

A mi madre Mariana Miranda† por ser fuente de inspiración y de revitalización de vida.

A mis hijos Sofía y Pedro, a mi nieta adorada Keyla, mis grandes sacrificados; pronto serán recompensados.

A Jorge Luis, mi hermano, por sus palabras de aliento que fueron un aliciente para poder mantener en el esfuerzo característico de nuestra familia.

AGRADECIMIENTO

A la Escuela Superior Politécnica de Chimborazo (ESPOCH), mi *alma mater*, por su apoyo invaluable para poder cursar los estudios doctorales.

Al pueblo peruano y a la Universidad Nacional Mayor de San Marcos, Decana de América, por la oportunidad de cursar en sus aulas el camino de la sabiduría, la ciencia y la investigación.

Al claustro docente de investigación y al personal administrativo del programa doctoral de la Facultad de Ingeniería de Sistemas e Informática (FISI), en la que tuve la fortuna de formar parte, gracias por su apoyo y dedicación.

A mi entrañable amigo Dr. Omar Gómez, por su experiencia y colaboración en determinadas etapas del programa doctoral.

A mi Tribunal de Sustentación, por sus recomendaciones, con las cuales este trabajo llegó a ser formulado de manera doctoral.

Una especial consideración y gratitud al Dr. Glen Rodríguez, mi supervisor y consejero, mi compañero de semana a semana durante estos seis años de travesía doctoral, sin su sapiencia y su visión de investigador no hubiese podido llegar a culminar este sueño académico.

A todos, un imperecedero sentimiento de gratitud.

ÍNDICE GENERAL

DEDICATORIA.....	III
AGRADECIMIENTO	IV
ÍNDICE GENERAL	V
LISTADO DE TABLAS	VIII
LISTADO DE FIGURAS	IX
LISTADO DE ECUACIONES.....	X
RESUMEN.....	XI
SUMMARY	XII
CAPÍTULO I: INTRODUCCIÓN	13
1.1. SITUACIÓN PROBLÉMICA	18
1.2. FORMULACIÓN DEL PROBLEMA.....	22
1.2.1. <i>Problema General</i>	22
1.2.2. <i>Problemas Específicos</i>	22
1.3. JUSTIFICACIÓN TEÓRICA	23
1.4. JUSTIFICACIÓN PRÁCTICA.....	26
1.5. OBJETIVOS	28
1.5.1. <i>General</i>	28
1.5.2. <i>Específicos</i>	28
CAPÍTULO II: MARCO TEÓRICO	29
2.1. MARCO EPISTEMOLÓGICO	29
2.2. ANTECEDENTES.....	32
2.3. METODOLOGÍA PARA LA REVISIÓN DE LA LITERATURA.....	32
2.3.1. <i>Método de investigación</i>	33
2.3.2. <i>Fuentes de información</i>	34
2.3.3. <i>Criterio de búsqueda</i>	35
2.3.4. <i>Selección de artículos</i>	35
2.3.5. <i>Extracción de datos y síntesis</i>	35
2.3.6. <i>Factores para la evaluación de los resultados</i>	39
2.3.7. <i>Resultados</i>	40
2.3.8. <i>Conclusiones</i>	43
2.4. BASES TEÓRICAS	45
2.4.1. <i>Técnicas de pruebas de regresión de software</i>	45
2.4.2. <i>Ingeniería de software con acceso a datos</i>	58
2.4.3. <i>Técnicas de pruebas de regresión en software con acceso a datos</i>	63
2.4.4. <i>Técnicas de clusterización para pruebas de regresión en software con acceso a datos</i>	65
CAPÍTULO III: METODOLOGÍA DE INVESTIGACIÓN.....	68
3.1. HIPÓTESIS	69
3.1.1. <i>Hipótesis General</i>	69
3.1.2. <i>Hipótesis Específicas</i>	69

3.2.	IDENTIFICACIÓN DE VARIABLES.....	70
3.2.1.	<i>Variable Independiente</i>	70
3.2.2.	<i>Variable Dependiente</i>	70
3.3.	OPERACIONALIZACIÓN DE VARIABLES.....	71
3.4.	MATRIZ DE CONSISTENCIA.....	72
3.5.	METODOLOGÍA.....	73
3.5.1.	<i>Tipo y diseño de la investigación</i>	73
3.5.2.	<i>Unidades de análisis</i>	74
3.5.3.	<i>Población de estudio</i>	74
3.5.4.	<i>Tamaño de la muestra</i>	75
3.5.5.	<i>Técnica de recolección de datos</i>	75
3.5.6.	<i>Métodos y técnicas de procesamiento y análisis de datos</i>	76
CAPÍTULO IV: EL APORTE.....		77
4.1.	MODELO CONCEPTUAL DEL APORTE.....	77
4.2.	DETERMINACIÓN DE REQUERIMIENTOS NO SATISFECHOS.....	80
4.3.	MÉTODO PARA PRUEBAS DE REGRESIÓN EN SOFTWARE CON ACCESO A DATOS	80
4.3.1.	<i>El enfoque</i>	80
4.3.2.	<i>Algoritmos diseñados para el método</i>	84
4.4.	SUPUESTOS.....	90
4.5.	ALCANCE.....	91
4.6.	LIMITACIONES.....	91
CAPÍTULO V: ESTUDIO EMPÍRICO		92
5.1.	DEFINICIÓN Y DISEÑO	93
5.2.	PROCEDIMIENTO Y PROTOCOLOS	94
5.2.1.	<i>Configuración</i>	94
5.2.2.	<i>Variables e indicadores</i>	96
5.2.3.	<i>Unidades de análisis</i>	98
5.3.	EJECUCIÓN Y RECOPIACIÓN DE EVIDENCIA	105
5.3.1.	<i>Planes de pruebas</i>	105
5.3.2.	<i>Instrumentación adicional</i>	106
5.3.3.	<i>Recopilación de evidencia</i>	108
CAPÍTULO VI: ANÁLISIS DE DATOS.....		114
6.1.	ANÁLISIS DEL IMPACTO DE FALLOS EN LAS PRUEBAS DE REGRESIÓN	114
6.2.	ANÁLISIS DE LAS PRUEBAS DE REGRESIÓN POR PRODUCTO DE SOFTWARE.....	118
6.3.	ANÁLISIS DE DATOS POR INDICADOR	124
6.4.	AMENAZAS A LA VALIDEZ	128
6.5.	CONTRASTACIÓN DE LAS HIPÓTESIS	129
6.5.1.	<i>Pruebas de normalidad a los datos de los grupos por indicador</i>	129
6.5.2.	<i>Hipótesis Específica H1: Cantidad de casos de prueba</i>	133
6.5.3.	<i>Hipótesis Específica H2: Precision</i>	135
6.5.4.	<i>Hipótesis Específica H3: Rendimiento combinado</i>	137
6.5.5.	<i>Hipótesis general</i>	140
6.6.	REPORTE.....	142
CAPÍTULO VII: CONCLUSIONES Y TRABAJOS FUTUROS		145
7.1.	CONCLUSIONES	145
7.2.	TRABAJOS FUTUROS	148

BIBLIOGRAFÍA	149
ANEXO A: PUBLICACIONES RESULTADOS DE LA TESIS	169
ANEXO B: PLAN DE PRUEBAS: SOFTWARE ACADÉMICO	170
ANEXO C: ANALISIS DEL IMPACTO DE FALLOS EN LAS PRUEBAS DE REGRESIÓN.....	185

LISTADO DE TABLAS

Tabla 1. Evolución del interés del estudio de los Sistemas de Información	20
Tabla 2. Bancos de datos electrónicos.....	34
Tabla 3. Técnicas detectadas en el estudio.....	37
Tabla 4. Lista de artículos de la fase de análisis y discusión.....	38
Tabla 5. Métricas modelo de eficiencia	39
Tabla 6. Operacionalización de variables.....	71
Tabla 7. Matriz de Consistencia	72
Tabla 8. Instrumentos para registro de métricas	76
Tabla 9. Resumen de técnicas de clusterización para software con acceso a datos.....	67
Tabla 10. Matriz casos de prueba con acceso a datos	81
Tabla 11. Identificadores usados en los algoritmos	84
Tabla 12. Metodologías y técnicas.....	95
Tabla 13. Características del producto software Estafeta	100
Tabla 14. Características del producto software Silabo	101
Tabla 15. Características del producto software Académico	102
Tabla 16. Características del producto software GICO	103
Tabla 17. Características del producto software Yellou	104
Tabla 18. Número de casos de prueba por versiones en cada producto software	109
Tabla 19. Casos de prueba que fallan en cada versión de producto software	110
Tabla 20. Clusterización versión 4 de Estafeta con varias semillas.....	112
Tabla 21. Clusterización versión 7 de Silabo con varias semillas.....	112
Tabla 22. Clusterización versión 5 de Académico con varias semillas	112
Tabla 23. Clusterización versión 5 de GICO con varias semillas	113
Tabla 24. Clusterización versión 3 de Yellou con varias semillas	113
Tabla 25. Comparativo global de las técnicas en el producto software Estafeta	114
Tabla 26. Comparativo global de las técnicas en el producto software Silabo	115
Tabla 27. Comparativo global de las técnicas en el producto software Académico	116
Tabla 28. Comparativo global de los métodos en el producto GICO	116
Tabla 29. Comparativo global de las técnicas en el producto software YELLOU.....	117
Tabla 30. Estadígrafos Resumen de cada producto software	118
Tabla 31. Estadígrafos de cada versión del software Estafeta.....	119
Tabla 32. Estadígrafos de cada versión del software Silabo.....	120
Tabla 33. Estadígrafos de cada versión del software Académico	121
Tabla 34. Estadígrafos de cada versión del software GICO	122
Tabla 35. Estadígrafos de cada versión del software Yellou.....	123
Tabla 36. Estadígrafos Resumen de cada producto software	124
Tabla 37. Indicadores por pruebas de regresión y grupos de análisis	124
Tabla 38. Comportamiento de los indicadores por grupos de análisis.....	140
Tabla 39. Valores de p de T Student en la contrastación de hipótesis	140
Tabla 40. Comportamiento de los indicadores por grupos de análisis.....	144

LISTADO DE FIGURAS

Figura 1. Diagrama conceptual del aporte.....	79
Figura 2. Método para pruebas de regresión en software con acceso a datos	83
Figura 3. Cantidad de casos de prueba por prueba de regresión y grupos de análisis	125
Figura 4. Precisión por prueba de regresión y grupos de análisis	126
Figura 5. Rendimiento combinado por prueba de regresión y grupos de análisis.....	127
Figura 6. Test normalidad: Cantidad de casos de prueba, grupo de control	130
Figura 7. Test normalidad: Cantidad de casos de prueba, grupo experimental.....	130
Figura 8. Test normalidad: Precision, grupo experimental.....	131
Figura 9. Test normalidad: Precision, grupo de control.....	131
Figura 10. Test normalidad: Rendimiento combinado, grupo experimental.....	132
Figura 11. Test normalidad: Rendimiento combinado, grupo de control	132
Figura 12. T Student: Cantidad de casos de prueba	134
Figura 13. Cantidad de casos de prueba por grupo de análisis.....	135
Figura 14. T Studente: Precisión por grupo de análisis	136
Figura 15. Precisión: Grupos de análisis	137
Figura 16. T student: Rendimiento combinado	138
Figura 17. Rendimiento combinado por grupos de análisis	139

LISTADO DE ECUACIONES

Ecuación 1. Indicador: Cantidad de casos de prueba	Error! Bookmark not defined.
Ecuación 2. Indicador: Precisión.....	97
Ecuación 3 Indicador: Recuperación	97
Ecuación 4. Indicador: Rendimiento combinado	97

RESUMEN

Actualmente, en diferentes contextos, en mayor o menor grado, los productos de software tienen acceso a bases de datos, siendo necesario considerar las pruebas de regresión para productos de software, tales como los sistemas de información. Esta tesis presenta dos aportes de investigación que permiten motivar e introducir una nueva técnica/método para pruebas de regresión por selección para software con acceso a datos.

El primer aporte establece los aspectos y factores considerados hasta el año 2020, que evidencia la escasa aplicabilidad de las investigaciones de la academia en la industria. El segundo aporte es el diseño de un método para pruebas de regresión que realiza una combinación de técnicas de clusterización probabilística no supervisada con valores de centroide aleatorios y pruebas unitarias, que juntamente con el esquema de la base de datos, determinar los casos de prueba relacionados con las modificaciones o adiciones de nuevas funcionalidades del producto software. El método fue validado empíricamente bajo el enfoque de ingeniería de software experimental, mediante 32 de 37 pruebas de regresión de cinco proyectos de software de pequeña y mediana escala, utilizando indicadores del campo de la búsqueda y recuperación de información. Los resultados sugieren que el método propuesto mejora las pruebas de regresión en productos de software con acceso a datos bajo entornos de desarrollo iterativo incremental.

Palabras claves: pruebas de regresión por selección, clusterización no supervisada, clusterización probabilística, pruebas unitarias, esquema de bases de datos, eficiencia, cantidad de casos de prueba, capacidad de reducción, precisión, recuperación, capacidad de detección, rendimiento combinado, f-measure

SUMMARY

Currently, different contexts to a greater or lesser degree, software products have access to databases, and it is necessary to consider regression tests for software products, such as information systems. This thesis presents two research contributions that allow to motivate and introduce a new technique for regression tests by selection for software with access to data.

The first contribution establishes the aspects and factors considered up to the year 2020, and it is the evidence the scant applicability of the academy's research in the industry. The second is the design of a method for regression tests that performs a combination of unsupervised probabilistic clustering techniques with random centroid values and unit tests in conjunction with the database schema. This is to determine the test cases related to the modifications or additions of new functionalities of the software product. The method was validated empirically under the experimental software engineering approach, through 32 of 37 regression tests with five projects of small and medium scale analysis, using metrics in the field of searching and information retrieving of information. The results suggest that the proposed method improves the regression tests in software products with data access under incremental iterative development environments.

Keywords: regression tests by selection, unsupervised clustering, probabilistic clustering, unit tests, database schema, efficiency, reduction number of the tests cases, capacity, precision, recovery, detection capacity, combined performance, f-measure

CAPÍTULO I: INTRODUCCIÓN

Hoy en día, el software, juega un rol primario integral en el quehacer de la humanidad, de tal manera que, no hay casi ninguna actividad en la que no esté involucrada y aún más su dependencia se va incrementando, siendo relevante que el software se comporte de acuerdo con sus especificaciones para garantizar confiabilidad en su uso (Beizer, 1990).

Según el estándar IEEE 610.12, la ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, lo que implica la aplicación de ingeniería al producto software. Laplante (2007) lo define como la aplicación de un enfoque sistémico a las actividades de análisis y diseño, evaluación, implementación, pruebas, mantenimiento y reingeniería de software. En términos generales, todo proceso de desarrollo se compone de actividades o fases de análisis, diseño, codificación, pruebas y mantenimiento de software. (Pressman, 2010)

Una de las actividades cuyo interés investigativo se ha venido incrementando por el énfasis en el costo por lograr productos de calidad de software, según Hetzel y Hetzel (1991) es la fase de pruebas, la cual garantiza la calidad del software (Bertolino, 2007; Saha, 2008; Harrold, 1999) antes de la entrega al usuario final, que inclusive puede llegar a costar cerca del 50% del costo del proyecto (Myers, Sandler y Badgett, 2011).

En dependencia de la metodología de desarrollo de software, la fase de pruebas se ejecuta en diferentes etapas/fases del ciclo de vida o forma parte integral del proceso, así, en el modelo tradicional (cascada), la etapa de pruebas se considera luego de la codificación, y en otros forma parte integral del proceso (iterativo-incremental), tomando el nombre de pruebas unitarias, pruebas integración, pruebas de funcionalidad, pruebas de aceptación y pruebas de regresión, según el modelo de validación y verificación o modelo. V&V (Wallace y Fujii, 1989)

El mantenimiento del software, en diferentes etapas/fases, se enfoca con las modificaciones del producto, luego que este ha sido liberado y entregado al usuario final. Según el estándar 1219-1993 de la IEEE, el mantenimiento del software se define como la modificación del producto software después de la entrega, para corregir fallas, para mejorar su desempeño o adaptar al entorno que ha sido modificado; bajo este enfoque, cada vez que ocurre una modificación al software es necesario realizar pruebas que validen esas modificaciones contra el código previamente validado, a ello se denomina pruebas de regresión. (Rothermel y Harrold, 1993)

Un especial análisis se merece las pruebas de regresión (RT por sus siglas en inglés), cuya finalidad es detectar fallas en modificaciones al producto software que, por lo general, se presenta en etapas de mantenimiento o en ciclos cortos de desarrollo.

La finalidad de las pruebas de regresión es garantizar que las modificaciones o incrementos realizados no afecten funcionalidad validadas previamente del producto. Este proceso de ejecutar las pruebas ante cada cambio o modificación tiende a ser una actividad costosa por los factores involucrados, en el caso de realizarse manualmente: el tiempo y recursos

humanos especializados, y en el caso de automatización: el tiempo e infraestructura involucrada (hardware y software); siendo ello más sensible en proyectos de mediana y gran escala, por la cantidad de pruebas. El caso más elemental es la de aplicar todas las pruebas, siendo necesario aplicar alguna estrategia que permita reducir costos.

Las investigaciones y trabajos en RT, según Rosero, Gómez, y Rodríguez (2016), Engström, Runeson y Skoglund (2010), indican que esta problemática se la puede enfrentar desde varios enfoques, los cuales están directamente relacionados según se determinen las pruebas que se vayan a utilizar en una prueba de regresión.

La primera estrategia se denomina técnica de pruebas de regresión por minimización (RTM, por sus siglas en inglés), en este enfoque se consideran aquellas pruebas que no son redundantes, ambiguas u obsoletas. La segunda estrategia se denomina técnica de pruebas de regresión por priorización (RTP, por sus siglas en inglés), en este enfoque se asigna una determinada prioridad a cada prueba y con base en ello se aplica las pruebas que mayor prioridad tengan. La tercera estrategia se denomina pruebas de regresión por selección (RTS, por sus siglas en inglés), según este enfoque se determina las pruebas que estén directamente relacionadas con las modificaciones realizadas al producto software. Finalmente se encuentran las pruebas de regresión por optimización (RTO, por sus siglas en inglés), en la que se aplica técnicas de softcomputing, en esta estrategia se pretende determinar el subconjunto de pruebas de software utilizando técnicas de inteligencia artificial como es el caso del aprendizaje de máquina-clusterización (machine learning).

Desde esta perspectiva, en el año 2012, Yoo y Harman (2012) realizaron un amplio estudio sobre las diversas técnicas que se han aplicado

para pruebas de regresión, mostrando que las han aplicado bajo la estrategia de caja blanca con el objetivo de analizar la estructura del código del programa y la cobertura de los cambios en el producto software; cuando el objetivo se cambió al nivel de verificar su funcionalidad, se enfocó bajo estrategia de caja negra; el estudio también muestra que estas estrategias no son apropiadas cuando el nivel de escalabilidad de la aplicación aumenta, por lo que se prefiere verificar la funcionalidad basado en especificaciones antes que código fuente.

A partir del año 2000, la ingeniería de software cuenta con una iniciativa proveniente de la industria, así, se ha popularizado una serie de prácticas y recomendaciones que se las resume en el documento denominado *Manifiesto Ágil* («Agile Manifesto for Software Development», 2015), que propone, entre otras cosas, mejorar la los plazos de entrega y calidad del producto; algunas de estas recomendaciones se han plasmado en metodologías y técnicas denominadas ágiles, ejemplo de ello es la Programación Extrema (XP), que se fundamenta en el desarrollo guiado por las pruebas (TDD, por sus siglas en inglés), la integración continua (CI, por sus siglas en inglés) y refactorización (Refactoring), realizadas en ciclos (iteraciones) cortos de desarrollo llamados sprints. (Ficco, Pietrantuono, y Russo, 2011)

En la metodología XP/TDD (Sangwan & LaPlante, 2006), el desarrollador primero escribe un caso prueba, posteriormente el código que ese caso de prueba verifica y luego como un acto final del ciclo (historia de usuario) se realiza una depuración o limpieza del código usando principios de refactorización, en estos ciclos de desarrollo cortos, los desarrolladores al final han generado código limpio y casos de prueba que se deben integrar para lograr funcionalidades mayores, pudiendo entonces empezar la etapa de integración, donde se ejecutan pruebas de regresión automatizadas, a fin de integrar las funcionalidad programadas; este enfoque de ir integrando código y casos de prueba diariamente constituye la fase de integración continua,

siendo necesario validar las integraciones como modificaciones a funcionalidades anteriores a través de pruebas de regresión continuas e incrementales, este es el momento donde las técnicas de regresión cobran importancia de tal manera que el desarrollador tenga una retroalimentación lo más pronto posible si su código (funcionalidad) está afectando anteriores funcionalidades. (Cibulski y Yehudai, 2011)

Actualmente, en mayor o menor grado, los productos software tienen acceso a bases de datos, por lo que también es necesario considerar las pruebas de regresión en este contexto, Haraty, Mansour, y Daou (2001), Willmor y Embury (2005) y Rogstad, Briand, y Torkar (2013), dada la evolución cambiante de los datos, presentándose una situación en la que además de considerar al código hay que considerar a los datos, siendo más relevante cuando los datos están almacenados de acuerdo a un modelo de datos que fue considerado en el diseño, situación que posibilita que una sola instrucción considere un gran volumen de datos que puede afectar al comportamiento del producto software con acceso a datos.

A pesar de la gran cantidad de investigaciones en la línea de proponer nuevas técnicas de regresión, inclusive para diferentes tipos de aplicaciones Biswas, Mall, Satpathy, y Sukumaran (2009) –aplicaciones embebidas, aplicaciones de bases de datos– pocas son las que se han aplicado en ambientes industriales, destacan Rosero, Gómez, y Rodríguez (2016), pues la mayoría de estas han sido validadas en ambientes académicos y muy escasos para el entorno ágil Rafaqut Kazmi, Dayang N. A. Jawawi, Radziah Mohamad y Imran Ghani (2017); por lo que resulta muy importante determinar el cómo aplicar las técnicas de las pruebas de regresión, en particular, para productos software con acceso a datos bajo un entorno de desarrollo iterativo incremental en un ambiente industrial. (Arora y Bhatia, 2018)

1.1. Situación Problemática

Las pruebas de software, ya sea se los aplique bajo en el enfoque de desarrollo o bajo en el enfoque de mantenimiento de software, son una actividad costosa pero esencial y son la clave tecnológica para evaluar cuantitativamente la calidad del producto software, otorgándole confiabilidad al contar con un producto lo más posible libre de fallos (Leung y White, 1991).

De manera general, cada vez que un producto software es modificado, ya sea por su propia evolución dentro del desarrollo del proceso software o en su etapa mantenimiento, una prueba de regresión será necesario aplicar para validar aquellas modificaciones de tal manera que se garantice que no se han inyectado fallos inesperados en versiones anteriores y funcionales del producto software. (Rothermel y Harrold, 1997)

Las pruebas de regresión de software son diferentes de otras pruebas, porque un conjunto de casos de prueba ha sido construido para validar y verificar el producto software (Leung & White, 1989); Con respecto al proceso de ejecución de una prueba de regresión, la técnica básica es ejecutar todos los casos de prueba que conforman el conjunto de casos de prueba, lo que se denomina un prueba de regresión total (*retestALL*, en inglés), otra alternativa es determinar un subconjunto de casos de prueba, aquellos que se relacionen con las secciones modificadas/incrementadas al producto software (Pravin, 2013), este método de reducción podría ser válido siempre y cuando los costos combinados al determinar el conjunto de casos de prueba y su ejecución sean menores que el costo de ejecutar todos los casos de prueba, es decir, que el subconjunto mantenga la efectividad de un regresión total. (Di Nardo et al., 2015)

Según los autores Kumar, Sharma, y Kumar (2014), en proyectos de software de mediana y gran escala, la cantidad de casos de pruebas puede llegar a miles y cientos de miles, por lo que ejecutar una prueba regresión, con las técnicas de regresión tradicionales analizadas a la fecha, se convierte en una actividad prohibitiva por los costos y esfuerzos involucrados, siendo necesario determinar cómo reducir de manera efectiva el conjunto de casos de prueba de tal manera que se pueda reducir los costos y los esfuerzos involucrados.

De acuerdo a Rosero et al. (2016), existe una falta de aplicabilidad de las técnicas de pruebas de regresión en productos de software en contextos de la industria, ya sea porque fueron desarrollados con tecnologías antiguas, productos software de cientos de líneas de código (productos software de pequeña escala) y, por lo general, en el ámbito de la investigación académica, con ningún o escasos casos de prueba. Un ejemplo de ello es la escasa utilización de estas técnicas de regresión mediante frameworks de automatización como JUnit en productos software en la industria.

Según la conceptualización de los sistemas de información (Tabla 1) con enfoque de tecnología de información, los autores Bentley L. D., Dittman, K. C., y Whitten (2000) consideran que los sistemas de información de mayor popularidad en la industria son los sistemas de información de corte administrativo/organizacional (MIS) (Rogstad, Briand, Dalberg, Rynning, y Arisholm, 2011; Petersen, Feldt, Mujtaba, y Mattsson, 2008; Webster & Watson, 2002).

Por otro lado, O'Brien & George Marakas (2010) presentan una clasificación conceptual de los sistemas de información, ya sea considerándolo como soporte a las operaciones del negocio o como soporte

a la gestión administración, en este último se considera una subclasificación: sistemas para soporte a decisiones (DSS), sistemas para información ejecutiva (EIS), sistemas de procesamiento especializado (EPS) y sistemas información para la gestión (MIS).

Según Ajayi et al. (2007), se considera a la Academia como una organización en la se maneja un gran volumen de datos (estudiantes, profesores, administradores) y que se hace uso de sistemas de información orientado a la gestión y a la toma de decisiones.

Con el aparecimiento del agisismo, según Hummel (2014) Vidgen & Wang (2009); Haftmann et al. (2007); Chays et al. (2004), los desarrollos de productos de software del tipo sistemas de información en entornos de desarrollo iterativo incremental han cobrado relevancia en los últimos años.

Tabla 1. Evolución del interés del estudio de los Sistemas de Información

(Bentley L. D., Dittman, K. C., & Whitten, 2000)	Systems anlysis and Design Methods
(Webster & Watson, 2002)	Systematic mapping studies in software engineering
(Chays et al., 2004)	An AGENDA for testing relational database applications
Haftmann, Kossmann, & Lo, 2007	A framework for efficient regression tests on database applications
(Petersen et al., 2008)	Systematic Mapping Studies in Software Engineering
(Vidgen & Wang, 2009)	Coevolving Systems and the Organization of Agile Software Development
(Hummel, 2014)	State-of-the-Art: A Systematic Literature Review on Agile Information Systems Development
(Erik Rogstad & Briand, 2016)	Cost-effective strategies for the regression testing of database applications: Case study and lessons learned
(O'Brien & George Marakas, 2010)	Management Information Systems

Considerando en entormo global de la industria de software, se espera un crecimiento de 3.8 billones de dólares en el 2016, de los cuales el 28% es en gasto empresarial. El rubro gasto mundial de productos software es del 5.2% en el periodo 2005-2011, donde el gasto de productos software es de 5.9% (Acebo, M., y Nuñez, A., 2016).

En este tipo de aplicaciones, realizar cambios es ineficiente, puesto que se requiere realizar pruebas de regresión por cada nueva versión o modificación que a ella se lo realice. Por ejemplo, la multinacional Microsoft gasta más del 50% del costo de desarrollo en pruebas de software, otro ejemplo es la empresa SAP que tiene un producto cuyo tiempo de desarrollo fue de 18 meses, de los cuales seis se gastó en pruebas de software.

Por otro lado, en los últimos años se han venido aplicando técnicas de inteligencia artificial, dentro del campo de la minería de datos, para enfocar la problemática con técnicas de recuperación de información, por ejemplo, las técnicas de clusterización, logrando algunas mejoras, dichos enfoques no han considerado estrategias combinadas de lógica de datos y pruebas unitarias (Zhang, Gao, Zhou, y Wang, 2017).

Los enfoques hasta la fecha consideran como técnica el análisis de clúster y a las métricas de costo y eficiencia para medir la bondad de la técnica. En este trabajo se hace referencia a la métrica de costos como la capacidad de reducción del número de casos de prueba (tasa de reducción de la técnica), que según los autores Leung & White (1991) y Z. Chen et al. (2011) se considera como métricas de eficiencia a la capacidad de la técnica para la recuperación de información, entendiendo como información la cantidad de casos de prueba recuperados, dentro de ellos las siguientes: precisión, recall (capacidad de detección de fallos) y f-measure. (Willmor y Embury, 2005; Zhang, Gao, Zhou, y Wang, 2017)

Por ello es relevante diseñar un método para realizar pruebas de regression, que mejore la eficiencia de los casos de prueba de software con acceso a datos, el cual esté relacionado con modificaciones y que enmarque en el contexto de desarrollo iterativo incremental, de tal manera que permita

al desarrollador recibir una retroalimentación inmediata de las afectaciones de su código a funcionalidades anteriores del sistema.

1.2. Formulación del Problema

1.2.1. Problema General

¿De qué manera el método de clusterización probabilística mejora de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental?

1.2.2. Problemas Específicos

En este trabajo se considera los indicadores de cantidad de casos prueba y de búsqueda y recuperación de información: precisión, recuperación y rendimiento combinado, así entonces se plantea los siguientes problemas específicos.

- a. ¿De qué manera el método de clusterización probabilística reduce la cantidad de casos de prueba en las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental?
- b. ¿En qué manera el método de clusterización probabilística mejora la Precisión de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental?

- c. ¿En qué manera el método de clusterización mejora el rendimiento cambiando de las pruebas regresión de software con acceso a datos en entornos de desarrollo iterativo incremental?

1.3. Justificación Teórica

El objetivo principal de las pruebas de regresión de software es asegurar que las modificaciones realizadas al producto software, ya sea en etapa de mantenimiento o en desarrollo, no hayan inyectado fallos en funcionalidades anteriores (Rothermel y Harrold, 1993).

Las pruebas de regresión juegan un rol integral para mantener la calidad en futuras versiones del producto software, pero es una actividad costosa, según los autores: Leung & White (1991), Rothermel y Harrold (1993) y Rosero et al. (2016), en etapas de producción o de mantenimiento.

Este tópico ha sido objeto de amplios estudios y debates básicamente en la dirección de incrementar la eficiencia de este proceso. El mayor esfuerzo de los estudios ha sido analizar y evaluar las diferentes técnicas para seleccionar, minimizar, priorizar y últimamente optimizar los casos de prueba que serán aplicados en una prueba de regresión, ya sea en la etapa de desarrollo o de mantenimiento. Sin embargo, revisiones de la literatura que se han venido realizando muestran que existe una muy reducida aplicación de esta tecnología en el sector industria.I (Engström, Runeson y Skoglund, 2010; Yoo y Harman, 2012; Rosero et al., 2016)

A fin de generar un cuerpo de conocimiento en ingeniería de software, se ha aplicado metodologías de ingeniería de software experimental, como las publicaciones de experimentos controlados de Tuya et al. (2008); Biswas, Mall, SatPathy y Sukumaran (2009); Ficco y Russo (2011); Pravin (2013) y unos pocos han sido realizados en entornos de la vida real o en sistemas de gran escala (Wikstrand et al., 2009; Rogstad et al., 2011).

Existen revisiones de la literatura como por ejemplo Yoo & Harman (2012) y Rosero et al. (2016) cuyos resultados muestran que la mayoría de las investigaciones se enfocan en pruebas de caja blanca utilizando métricas de cobertura de código y utilizando diferentes estrategias como el análisis estructural de programas. Cuando el objetivo de la prueba es a nivel funcional, el estudio muestra que los evaluadores se orientan hacia enfoques de pruebas de caja negra basada en las especificaciones del producto software. Otros trabajos (*A Comparative Study of Clustering Deviations for Black-Box Regression Test - Selection Techniques - ProQuest*) también muestran que existen estudios que, además de contar con los enfoques de minimización, selección o priorización de casos de prueba, actualmente se está trabajando en la línea de las técnicas de optimización o del softcomputing, como por ejemplo el aprendizaje de máquina, redes neuronales, entre otras. (Yoo, Harman, Tonella, & Susi, 2009; Maia, do Carmo, de Freitas, Gomes, de Campos, de Souza, et al., 2010; Kumar, Sharma, & Kumar, 2012)

Dentro del tema de la diversidad de tipos de productos software, existe uno que ha sido objeto de menos estudio, según los autores Rogstad y Briand (2015), es el caso de los productos software que tienen acceso a base de datos que brindan un inmenso contingente de procesamiento de datos a las organizaciones. Tales productos han sido desarrollados con tecnologías antiguas y con escasas o nulas consideraciones de pruebas, ya que todo su proceso de pruebas se las realizó en su proceso de construcción de manera

manual sin considerar a futuros aspectos de crecimiento de requerimientos o de procesos de mantenimiento en los cuales el proceso manual no es manejable, una manera práctica para enfrentar este problema fue la introducción de procesos de automatización de pruebas, principalmente para productos software que comienzan a escalar en tamaño y complejidad, siendo relevante mantener la sostenibilidad del producto software en las empresas que se encuentran inmersas en este contexto.

Lo anterior va en la estrategia de reducir el tamaño del conjunto de los casos de prueba que se aplicarían en una prueba de regresión, lo que implica la reducción de los esfuerzos mientras se maximiza la posibilidad de detectar las fallas de regresión y los casos de prueba relacionados con ellos. Una medida para asegurar el objetivo frente a una escalabilidad del tamaño del producto y manejar el esfuerzo manual al mínimo es agrupar y ejecutar los casos de prueba que estén relacionados con un fallo.

En el contexto de la evaluación de las propuestas se han venido utilizando métricas de los indicadores de costos y eficiencia ((Zhou et al., 2018). Dentro del indicador de costos se considera el tiempo necesario para realizar la prueba de regresión, el cual está directamente relacionado con la capacidad de determinar los casos de prueba que están relacionados con los fallos del producto software (Leung & White, 1991).

Para la métrica de eficiencia, es analizada desde la perspectiva del desempeño de las técnicas de recuperación de información, según Zhang, Zhou y Wang (2017), considerando como información a los casos de prueba. Las métricas en este campo son las siguientes: precisión (P) que evalúa la proporción de casos de prueba relevantes recuperados respecto al total de los casos de prueba recuperados del sistema, entendiéndose como casos de prueba relevantes aquellos que detectan fallos; recall (R) como la proporción

entre los casos de pruebas relevantes recuperados y los casos de prueba relevantes; en este campo, (P) y (R) son métricas inversamente proporcionales, siendo necesario encontrar un balance entre ellas, la métrica que considera este equilibrio es F-measure que es definida como la media armónica entre (P) y (R) (Kumar, Jandsalar, & Rawat, 2017).

Para reducir este vacío entre la academia y la industria en el tema de las pruebas de regresión de software, es necesario realizar estudios empíricos en entornos de desarrollo del mundo real que direccionen las investigaciones a la práctica y permita evaluar las soluciones en contextos de industriales y así ampliar el cuerpo de conocimiento de las pruebas de regresión para software con acceso a datos.

1.4. Justificación Práctica

Una investigación del 2002, realizada por el Departamento de Comercio y el Instituto Nacional de Normas y Tecnologías de los Estados Unidos, estimó un daño económico anual alrededor 59.5 billones de dólares, como resultado de una inadecuada infraestructura y proceso de pruebas de software; el artículo clasifica los daños en dos categorías: mal funcionamiento del software y modificaciones al software y reprueba al sistema, mencionando que existe una alarma acerca de que los costos involucrados en las pruebas de software son altos y ellos no se han reducido pese al pasar de los años.

A pesar de lo anterior, según un estudio de ACM (Wagner & Seifert, 2005), se hace notar que la actividad de pruebas es importante para conseguir la calidad del software y que puede redondear el 50% del presupuesto del proyecto.

Por otro lado, algunos estudios como los de Rothermel y Harrold, (1998), Yoo y Harman (2007), Engström, Runeson Skoglund (2010), Rosero et al.(2016), Kazmi, Jawawi, Mohamad, y Ghani (2017) y Arora y Bhatia (2018) muestran que los estudios de pruebas de regresión se ha ido incrementando, incluyendo enfoques actuales de desarrollo como los modelos ágiles y técnicas de softcomputing (Özkan, Garousi, & Betin-Can, 2017), algunos de ellos enfocándose en la industria (Rogstad, Briand y Arisholm, 2011); así como también considerando productos software que considera el acceso a datos (Rogstad y Briand, 2015; Zhang, Gao, Zhou, y Wang, 2017); siempre considerando una retroalimentación de los resultados de una prueba de regresión lo más cercano al desarrollador que se encuentre inmerso en proceso de desarrollo iterativo incremental. (Lonetti y Marchetti, 2018)

En la práctica constituye un reto el diseñar e implementar una alternativa de prueba de regresión para productos software con acceso datos por consideraciones propias de este tipo de productos, como por ejemplo el elevado y complejo número de consultas a la base de datos, la dificultad de contar de manera combinada una relación entre el código y los datos almacenados en una base de datos, es por ello que se ha considerado aplicar técnicas de softcomputing como aprendizaje de máquina que permita determinar modelos de comportamiento de datos y los relaciones con modificaciones con las pruebas que consideran acceso a la base datos.

Lo anterior posibilitará al sector de la industria impulsar de manera más asequible financieramente el hecho de aplicar procesos de validación y verificación de software de tal manera que mejore la calidad del producto, beneficiándose la sociedad en su conjunto al contar con productos software con garantía de calidad.

1.5. Objetivos

1.5.1. General

Diseñar un método de clusterización probabilística que mejore las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental

1.5.2. Específicos

Para alcanzar el objetivo general definido, es necesario alcanzar los siguientes objetivos específicos:

- a) Reducir de casos de prueba en las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental.
- b) Mejorar la precisión de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental.
- c) Mejorar el rendimiento cambiando de las pruebas regresión de software con acceso a datos en entornos de desarrollo iterativo incremental

CAPÍTULO II: MARCO TEÓRICO

2.1. Marco epistemológico

La epistemología como campo de la filosofía estudia el conocimiento, sus definiciones y conceptos relacionados, en esa dirección, desde la perspectiva más general, encontramos algunas definiciones sobre las pruebas de software que están relacionados con aspectos, actividades o con problemas específicos, así Myers, Sandler y badgett (2011) define las pruebas de software como el proceso de ejecutar un programa con el fin de hallar fallos. Hetzel y Hetzel (1991) define las pruebas de software como la actividad enfocada a evaluar un atributo o capacidad de un programa, luego entonces las pruebas de software permiten medir la calidad del producto. Según Yoo y Harman (2012), las pruebas de software juegan un rol importante en algunas etapas del ciclo de desarrollo de un producto de software o en etapas posteriores, puesto que verifican que no existan fallos en el caso de mantenimiento.

De otro lado, un tipo particular de prueba de software constituye las pruebas de regresión de software, son aquellas pruebas aplicadas al producto software cuando este es objeto de mantenimiento o incremento de nuevos requerimientos, cuyo propósito es verificar que no se hayan generados fallos en funcionalidades previamente verificadas (Rothermel y Harrold, 1993).

Del lado de la eficiencia, la ejecución del conjunto completo de casos de pruebas es una actividad costosa y laboriosa, por lo que la determinación de un subconjunto de casos de prueba que se deban ejecutar en una prueba de regresión de software es aún objeto de ingente investigación.

El tamaño del producto software y su conjunto de casos de prueba tiene una relación directa con el costo de una prueba de regresión, específicamente por la limitación de recursos computacionales (Leung y White, 1991; Di Nardo, Alshahwan, Briand & Labiche, 2015).

A la fecha se viene estudiando y aplicando diferentes técnicas que van desde enfoques de eliminación de test cases obsoletos o redundantes hasta la priorización de ellos o la selección para obtener el subconjunto, entre estas estrategias destacan las técnicas de minimización, de priorización, de selección, y recientemente optimización según Rosero et al. (2016); en particular, en optimización se aplican estrategias para la determinación del grupo de casos de prueba que son utilizados en una prueba de regresión como por ejemplo Heurísticas Greedy en Minimización (Parsa y Khalilian, 2009). Panigrahi y Mall (2012) trabajan utilizando relaciones de dependencia basada en grafos en selección. Por otro lado, basadas en su capacidad de detección de defectos en priorización esta el trabajo de Pravin (2013), y, por último, Basados en lógica fuzzy para la de optimización. (Kumar, Sharma y Kumar, 2014)

Si se considera entornos de desarrollo iterativo incremental para software de mediana y gran escala, la ejecución del conjunto total de casos de prueba es una estrategia inaceptable dado el volumen de ellos.

Recientemente se han venido investigando enfoques de softcomputing en entornos ágiles, como el de Rogers (2004) para la identificación de casos de prueba, de manera que ellos garanticen la confiabilidad de que no se han inyectado fallos en el sistema (Biswas, Mall, Satpathy y Sukumaran, 2009).

Resultados de estudios recientes de Podgurski y Yang (1993); Mohanty, Ravi, y Patra (2010) y Gökçe, Belli, Eminli, y Dinçer (2015), que indican que una alternativa efectiva es el agrupamiento o clusterización de test cases, de acuerdo a algún criterio, patrón o característica de casos de prueba así, se espera que los casos de prueba que detectan el mismo fallo estén en el mismo clúster, como por ejemplo perfiles de ejecución de pruebas de los autores Vangala, Czerwonka y Talluri (2009); Zhang, Chen, Zhao, Yan, y Xu, (2010); en históricos de ejecuciones de pruebas (Chen, Duan, Zhao, Xu y Qian., 2011); llamadas a funciones (Guo, Subramaniam, & Chundi, 2012), particionamiento de programas, filtración de particiones (Chen, Duan, Zhao, Xu, y Qian, 2011); acceso a bases de datos (Rogstad y Briand, 2015); muestreo de los más representativos casos de prueba de cada clúster (Yan et al., 2010). El reto investigativo en estos casos es determinar la cantidad y el tamaño de clústeres para balancear la relación costo y eficiencia. (Zhou, Sinaga, Susilo y Zhao, 2018)

De acuerdo a Zhang, Zhou y Chen (2007), específicamente en el ámbito de aprendizaje de máquina, el análisis de clúster es una técnica para entrenar y organizar datos según enfoques matemáticos independientes del dominio de aplicación; en algunas ocasiones se necesita contar con información adicional a los datos, como por ejemplo el número y tamaño de clústeres a conformar, lo que encarece y en ocasiones limita el procedimiento, por lo que la clusterización supervisada requiere de experiencia para agrupar dicha información (Bilenko et al., 2004); una alternativa es la clusterización no supervisada en el que los datos no estén etiquetados y criterios probabilísticos

son usados para determinar el número de clústeres. (Berkhin, 2006; Haraty, Mansour, Moukahal y Khalil, 2016)

En productos software con acceso a bases de datos, se observan dos tipos de estados simultáneamente (estado del código y estado de los datos), por lo que las técnicas de pruebas de regresión deben considerar estos dos aspectos; y es que mientras el estado del código está organizado como un conjunto de localidades de memoria etiquetados donde los datos individuales son almacenados, el estado de una base de datos está organizado de acuerdo a diferentes conceptos del modelo de datos, como por ejemplo el modelo relacional; por otro lado, grandes volúmenes de datos pueden verse afectados por una única instrucción del código, lo cual puede afectar el comportamiento de otras instrucciones en componentes separados del producto software que accede a una base datos. (Willmor y Embury, 2005)

2.2. Antecedentes

Para poder establecer una línea base para la investigación, se realizó un estudio del estado del arte, para lo cual se aplicó una revisión sistemática de la literatura a los trabajos investigativos de los últimos 18 años (2000 – 2017), enmarcando en la taxonomía de las técnicas previamente descritas trabajos publicados en las revistas con factor de impacto (Rosero & Rodríguez, 2016; Rafaqut Kazmi, Dayang N. A. Jawawi, Radziah Mohamad, and Imran Ghani, 2017; Lonetti & Marchetti, 2018; Arora & Bhatia, 2018).

2.3. Metodología para la revisión de la literatura

Dado el carácter y el dominio de la investigación, es decir investigativo y de ingeniería de software, se aplicó la metodología de revisiones

sistemáticas, metodología propuesta por Kitchenham, Brereton, Budgen y Bailey (2009) para realizar análisis cualitativo de los resultados de las investigaciones en el lapso de tiempo y la taxonomía de técnicas de regresión (Minimización, Priorización, Selección, Optimización).

2.3.1. Método de investigación

Siguiendo la metodología que basa la recolección de la literatura de acuerdo en cuestiones de investigación y las respuestas a ellas, en este trabajo se consideró las siguientes cuestiones de investigación (CI):

- CI1: ¿Cuáles técnicas de pruebas de regresión han sido documentadas?
- CI2: ¿Que métricas e indicadores son los más utilizadas?
- CI3: ¿Cuáles son las estrategias implementadas en una prueba de regresión?
- CI4: ¿En qué contexto las pruebas de regresión han sido implementadas?
- CI5: ¿En qué fases del desarrollo del software han sido implementadas?

2.3.2. Fuentes de información

Según Kitchenham, Brereton, Budgen y Bailey (2009), la búsqueda de investigaciones se basa en la búsqueda en bancos de datos electrónicos, en este caso se utilizaron las siguientes fuentes:

Tabla 2. Bancos de datos electrónicos

Bases de Datos Electrónicas
• ACM Digital Library • IEEE Digital Library • HINDAWI Publishing Corporation (advances in software Engineering) • International Journal of Advanced Research in Computer Science and Software Engineering • ArXive preprint: 1108.4094 ArXiv.org Cornell University • Internacional Journal of Computer Science and Engineering IJSCE • Journal of computer Science • Elsevier ScienceDirect • CLEI Electron. J. 11(1) • Zoltan Csömyei (ed) 72 • Journal of Networks • International Journal of Computer Applications • Chiang Mai J. Sci. (ejournal) • International Journal of Software • Springer

Además, se tomó en cuenta conferencias, revistas y talleres con relación a pruebas de regresión de software, mantenimiento de software e ingeniería de software.

2.3.3. Criterio de búsqueda

Las palabras claves utilizadas en la estrategia de extracción de artículos basado en el título y resumen de artículo, en el periodo antes mencionado, y en idioma inglés.

<regression> AND <test OR testing> AND <selection>

< technical or method> AND < database>

La búsqueda inicial dio como resultado un universo de 470 artículos.

2.3.4. Selección de artículos

Para la selección de artículos se realizaron varias etapas de refinación:

Etapas 1: Análisis con base en el título (75)

Etapas 2: Análisis con base en el resumen (40)

Etapas 3: Análisis con base en el texto completo (30)

2.3.5. Extracción de datos y síntesis

Como resultado de la etapa anterior se obtuvo 30 artículos que consideraron 34 técnicas diferentes (ver Tabla 3), considerando como técnica el método que ha sido evaluado empíricamente mediante un caso de estudio o una evaluación experimental a un artefacto objeto de estudio en la investigación; con respecto al tamaño del artefacto, se ha considerado la

clasificación en pequeño mediano y grande que corresponde al número de líneas de código (LOC por sus siglas en inglés) de esas clasificaciones: pequeño hasta 10.000 LOC, grandes mayores a 50.000 LOC y mediano en el rango intermedio (ver Tabla 4).

Tabla 3. **Técnicas detectadas en el estudio**

T1	Detección de fallas basado en la información del estado de objetos en sistemas O. O.
T2	Detección de fallas baso en el modelo de representación de programas embebidos
T3	Particionamiento de programas (slicing)
T4	Basado en algoritmos genéticos
T5	Basado en información histórica almacenada en caché
T6	Detección de fallas considerando métrica de tiempo de los test
T7	Minería de datos
T8	Selección de prueba para cambios de código O.O a nivel de método
T9	Optimización multi Objetivo
T10	Lógica Difusa
T11	Selección de prueba basada en métricas de cobertura de flujo de datos y control
T12	Particionamiento, selección (dos fases)
T13	Visualización de Código
T14	Análisis continuo (replicación de código base)
T15	Extensión de Junit a Grid heterogéneo y distribuida
T16	Retest total
T17	Priorización de test basado en modelos de programas
T18	Heurísticas Greedy
T19	Basado en información de dependencia de datos, control y de estado (comportamiento)
T20	Análisis dinámico de programas usando la representación del programa en el IDE
T21	Selección de prueba basado en información de cobertura de flujo de control
T22	Comparación de comportamiento de programas a través de análisis dinámico
T23	Usando métricas de tiempo de ejecución
T24	Análisis dinámico de programas
T25	Lenguaje natural
T26	Integración Continua
T27	Grasp
T28	Simulated Annealing
T29	Programación GOAL
T30	MQHDE Multi Objetivo basada en técnica cuántica
T31	NSGA-II Algoritmo genético no ordenado
T32	Firewall para procedimientos almacenados
T33	Manejo de transacciones de base de datos usando grafos
T34	Desviaciones de estados de la base de datos

Tabla 4. Lista de artículos de la fase de análisis y discusión

Id	Enfoque	Técnica	Estrategia	Métrica	Tamaño	Contexto
T19	P	RTS	Información Histórica	Pr	L	I
T6, T20	C	RTS	Código	Pr	S	A
T4, T5	C	RTS	Modelos	DDE	S	A
T4, T6, T7	C	RTS	Código	TCSR	S	A
T22, T23	P	RTS	Modelos	Pr, DDE	M	A
T9, T17	P	RTS	Código	ETR	M	A
T21	P	RTS	Casos de prueba	DDE	B	A
T8	C	RTS	Información Histórica	DDE	M	A
T11, T12, T32	C	RTS	Inteligencia artificial	TCSR, ETR	M	A
T1, T33, T34	P	RTS	Inteligencia artificial	TCSR, ETR	M	A, I
T2, T3, T31	C	RTM	Heurísticas	TCSR, ETR	M	A
T15, T18	C	RTS	Código	Pr	M	A, I
T28, T29, T30	C	RTO	Heurísticas	TCSR, ETR	S, M	A
T15	P	RTS	Modelos	DDE	S, M, L	A
T13, T15	C	RTS	Código	Pr	M, L	A
T25, T27	C	RTP	Modelos	DDE	S	A
T6, T18, T19	C	RTS	Código	DDE	S	A
T24	C	RTS	Código	DDE	M	A
T1	C	RTM	Casos de prueba	DDE	S	A
T14, T16	C	RTS	Código	P, ETR	S	A
T26	P	RTP	Código	DDE	S, M, L	A
T4, T5	C	RTS	Código	DDE	S	A
T10	P	RTS	Código	Pr	M	I
T16	P	RTS	Código	Pr	S	A
T6, T8	C	RTS	Información Histórica	Pr	L	I
Enfoque	P= Progressive, C=Corretive					
Técnica/Método	RTS = Regression Test Selection RTP = Regression Test Prioritization RTM = Regression Test Minimization RTO = Regression Test Optimization					
Métrica/Indicador utilizada	TCSR = Test Case Suite Reduction ETR = Execution Time Reduction DDE = Defect Detection Efficiency Pr = Precisión					
Tamaño del sistema	S = Small, M = Medium, L = Large					
Contx. app	A= Académic, I =Industrial					

2.3.6. Factores para la evaluación de los resultados

Se ha utilizado el modelo de costos/beneficio propuesto por Do, Rothermel, y Kinneer (2006) bajo dos categorías de métricas: costos de reducción y eficiencia en la detección de fallas; se consideran tres aspectos en relación a reducción de costos y uno relacionado a la eficiencia en la detección de fallas.

Los factores claves de resumir son:

- Capacidad de reducción del conjunto de casos de prueba.
- Precisión (omisión de casos pruebas que no revelan fallas).
- Recall relacionado a la eficiencia en la detección de fallas de los casos de prueba.

Los mismos que se trabajan como métricas, según la Tabla 5.

Tabla 5. Métricas modelo de eficiencia

	Métrica	Definición
Eficiencia en la prueba de regresión	Capacidad de reducción del conjunto de casos de pruebas	Porcentaje del subconjunto de casos de prueba de la suite total (TR)
	Recuperación relacionada con la eficiencia detección de fallos del caso de prueba (recall)	Porcentaje de detección de fallas de los casos de prueba (R)
	Precisión de detectar el fallo (precisión)	Omisión de casos pruebas que no revelan fallas (P)
	Rendimiento combinado	Harmónica de global rendimiento entre R y P (RC)

2.3.7. Resultados

A continuación, se presenta un análisis de la información obtenida del estado del arte sobre las pruebas de regresión de productos de software.

Para la pregunta de investigación CI1, se identificaron 34 técnicas (T1... T34), las mismas que se listan en la Tabla 3, según la cual la técnica que predomina son las pruebas de regresión por selección, detectando que en los últimos años la tendencia incluye técnicas de optimización basadas en inteligencia artificial.

Con respecto a las métricas utilizadas (CI2), se observa que la capacidad de detección de fallas y la precisión de los casos de prueba son los temas de mayor interés, dejando las métricas relacionadas a la capacidad de reducción de casos de prueba y al tiempo de implementación en un segundo plano. 20 de 25 artículos identificados (80%) consideran las siguientes métricas:

Relativas a la eficiencia y a la capacidad de detección de fallas (Parsa, S. & Khalilian, A., 2009; Panigrahi, C. R. & Mall, R., 2012; Biswas et al., 2013; Kim, Zimmermann, Whitehead Jr., & Zeller, 2007; Ficco, Pietrantuono & Russo, 2011; Muşlu et al., 2013; Pravin, 2013; Panigrahi & Mall, 2010; Cibulski & Yehudai, 2011).

Relativas a la eficiencia y a la precisión (Orso et al., 2004; Rothermel & Harrold, 1997; Mirarab et al., 2012; Wikstrand, Feldt, Gorantla, Zhe & White,

2009; Soetens et al., 2013; Steinert et al., 2010; Anderson, Salem, & Do, 2014; Gladys Machado Pereira S y Lima, et al., 2008).

Seis artículos (24%) discuten reducción de costos (Kumar, M. et al., 2012; Kumari, A. C. et al., 2013; Duarte et al., 2006; Maia, do Carmo, de Freitas, Gomes, de Campos, de Souza, 2010).

Tres de los cuales Kumari et al. (2013) y Rothermel y Harrold (1997), son aplicados en ambientes de desarrollo iterativo incremental (12%), los cuales consideran la reducción del tiempo como un factor relevante para entornos incrementales de desarrollo para nuevas funcionalidades.

Respecto a la CI3, que se relaciona con las estrategias aplicadas en una prueba de regresión, se encontró que se encuentran basadas en las siguientes estrategias:

- código fuente, los autores: Panigrahi, C. R. & Mall, R. (2012), Ball, (1998), Soetens, Demeyer y Zaidman (2013), Steinert, Haupt, Kranh & Hirschfeld (2010), Pravin (2013), Gladys et al. (2008) y Rogers (2004).

- modelos de programas, los autores: Biswas, Mall, Satpathy & Sukumara (2009); Muşlu, Brun, Ernst y Notkin (2013) y Cibulski y Yehudai (2011).

- información históricas de ejecución de casos de prueba, los autores: Wikstrand, Feldt, Gorantla, Zhe, y White (2009); Kim, Zimmerman, Whitehead y Zeller (2007); Anderson, Salem y Do (2014) trataron este técnica.

- casos de prueba, los autores: Parsa y Khalilian (2009); (Mirarab, Akhlaghi y Tahvildari (2012) y Ficco, Pietrantuono y Russo (2011).

- basado en heurísticas, los autores: Kumari, Srinavas & Gupta (2013) ; Maia, do Carmo, de Freitas, Gomes, de Campos, y de Souza (2010).

- basados en enfoques de inteligencia artificial (Kumar, Shumar & Kumar, 2012).

Respecto a los contextos de la investigación (CI4), se observó que solo 4 investigaciones de: Wikstrand, Feldt, Gorontla, Zhe White (2009); (Soetens, Dmeyer & Zaidma (2013); Anderson, Salem, & Do (2014); Gladys Machado et al. (2008), fueron aplicadas en contexto industrial en los que se considera programas de pequeña y mediana escala, haciendo notar que los casos tratados consideran su código como propietario, por último, estos casos no han sido suficientemente evaluados para poder generalizar sus resultados, siendo en algunos casos extensiones de trabajos anteriores de Kumari, Srinavas & Gupta (2013); Panigrahi, C. R. & Mall, R. (2012); Biswas, Mall, Satpathy & Sukumara (2009), Biswas, Mall, & Satpathy (2013) y Kim, Zimmerman, Whitehead & Zeller (2007).

Finalmente, con respecto al CI5, revisando la tabla A1 en el apéndice se observa que las técnicas tradicionales son aplicadas tradicionalmente bajo enfoque de mantenimiento de software, es decir, con enfoques correctivos (64%) y principalmente en programas bajo el paradigma orientado a objetos o procedural.

2.3.8. Conclusiones

En la aplicación de la revisión sistemática de la literatura, para las pruebas de regresión de software se identificaron 34 técnicas que se enfocan en:

- adaptación de las pruebas de regresión a distintas áreas (Panigrahi, C. R. & Mall, R., 2012); (Rothermel & Harrold, 1997); (Biswas, Mall, & Satpathy, 2013); (Wikstrand et al., 2009); (Soetens, Dmeyer & Zaidman, 2013); (Pravin, 2013); (Panigrahi & Mall, 2010); (Gladys Machado Pereira S. Lima, et al., 2008).

- identificaron nuevos algoritmos (Kumar, Shumar & Kumar, 2012); (Parsa & Khalilian, 2009); (Maia, do Carmo, de Freitas, Gomes, de Campos, & de Souza, 2010).

- optimización de casos de prueba (Panigrahi, C. R. & Mall, R., 2012); (Rothermel & Harrold, 1993); (Ficco, Pietrantonio, & Russo, 2011); (Muşlu et al., 2013); (Steinert, Haupt, Kranh & Hirschfeld, 2010); (Cibulski & Yehudai, 2011).

- el uso de técnicas con enfoque de inteligencia artificial (Kumar, Shumar & Kumar, 2012); (Biswas, Mall, Satpathy, & Sukumaran, 2009); (Anderson, Salem & Do, 2014); (Cibulski & Yehudai, 2011).

Así también se observaron artículos que mencionan las pruebas de regresión en el contexto de desarrollo ágil (Ficco, Pietrantonio, & Russo,

2011); (Gamma & Beck, 2000); (Steinert, Haupt, Kranh & Hirschfeld, 2010): (Cibulski & Yehudai, 2011).

Con enfoque de inteligencia artificial (Kumar, Shumar & Kumar, 2012); (Parsa, S. & Khalilian, A., 2009); (Anderson, Salem & Do, 2014); (Maia, do Carmo, de Freitas, Gomes, de Campos, de Souza, 2010).

concluyendo que las pruebas de regresión aún se siguen aplicando en el contexto de mantenimiento de software y en contextos académicos; por lo que no se puede generalizar los estudios por la escasez de técnicas aplicadas en contexto industriales, debido a que industria invierte de manera reducida en procesos de verificación de software ya sea por en la complejidad del producto de software o por en la limitación de tiempo de los proyectos de software.

Sin embargo, se observa una fuerte tendencia a estudiar la eficiencia en la detección de fallos de las pruebas de regresión con enfoque de inteligencia artificial. (Kumar, Shumar & Kumar, 2012); (Biswas, Mall, Satpathy & Sukumara, 2009); (Anderson, Salem & Do, 2014); (Cibulski & Yehudai, 2011). Heurísticas de optimización (Kumar, Shumar & Kumar, 2012); (Parsa & Khalilian, 2009); (Maia, do Carmo, de Freitas, Gomes, de Campos, & de Souza, 2010) donde enfoque multi objetivos está empezando a utilizarse en la optimización del conjunto de casos de prueba con el objetivo de reducir los costos y mejorar su eficiencia.

En el contexto de software con acceso a datos, Haraty *et al.* (2001) consideraron las pruebas de regresión desde la perspectiva de procedimientos almacenados con enfoque de firewall. Willmor y Embury (2005) presentaron su técnica de prueba de regresión segura de acuerdo con

el enfoque de caja blanca usando representación de grafos para el código de las aplicaciones. Tuya *et al.* (2008) propusieron una serie de experimentos para evaluar y comparar la eficiencia de varias técnicas de caja blanca para instrucciones SQL (Structured Query Language) con el objetivo de seleccionar casos de prueba asociados a bases de datos. Rogstad *et al.* (2013) consideró los aspectos transaccionales de las bases de datos, pero desde la perspectiva de caja negra usando árboles de clasificación para representar similitudes de los casos de prueba. De manera similar, Rogstad y Briand (2016) consideraron casos de prueba relacionados a bases de datos, tratando las desviaciones o diferencias entre distintos momentos de una bases de datos, de acuerdo a los diferentes manipulaciones de los datos según las versiones de los programas; este enfoque está basado en criterios de entropía para evaluar diferentes de clusterización (Y.-F. Chen, Rosenblum, y Vo, 1994).

2.4. Bases Teóricas

El análisis y discusión de la literatura se realizó de acuerdo con la taxonomía considerada para la organización de las investigaciones, es decir, de acuerdo con las pruebas de regresión por selección, por minimización, priorización y optimización, en las cuales se evidencia las debilidades y fortalezas de las técnicas en relación con la reducción de costos y a la eficiencia en la detección de fallas.

2.4.1. Técnicas de pruebas de regresión de software

- ***Pruebas de regresión por minimización***

Se considera aquellos casos de pruebas que quedan en la suite de casos de prueba luego de una depuración de casos de prueba redundantes,

ambiguos, obsoletos, quedando la suite total con aquellos casos de prueba para aplicación de una prueba de regresión.

Formalmente, dado un conjunto de casos de prueba, T , un conjunto de requerimientos $r_1, r_2, \dots, r_n$, deben satisfacer para proveer una apropiada prueba a un programa y un subconjunto $T_1, T_2, \dots, T_n$, asociados con cada r_i , donde cualquier caso de prueba t_j perteneciente a T_i , puede ser usado para satisfacer los requerimientos de r_i . El problema se enfoca en determinar un representativo T' de casos de prueba, un subconjunto de T que satisfaga todos los r_i requerimientos de T . A continuación, las técnicas que se encuentran en esta categoría.

- *Basados en heurísticas*

(Kumari *et al.*, 2013) presentan una herramienta de apoyo para el grupo de testadores que permite reducir el conjunto de pruebas bajo un enfoque de Pareto, la herramienta denominada REGRESSAID (MQHDE) experimentalmente es comparada con otros enfoques de la línea Greedy y NSGAII, mostrando que la propuesta es superior en relación con la eficacia. REGRESSAID implementa dos algoritmos de optimización mutli objetivo: MQHDE propuesto anteriormente por Charan Kumari *et al.* (2013) y el segundo algoritmo NSGAII que sirven para tratar los problemas de optimización tipo Pareto; no se ha realizado evaluaciones en sistemas industriales de gran tamaño.

Saeed Panza y Alireza Khalilian (2009) proponen una técnica para reducir los casos de pruebas en una prueba de regresión, pero ellos consideran la contribución de un algoritmo Greedy Bi objetivo (maximizando las características de requisitos de un TC y minimizando la reducción entre TC) tratando problemas como de optimización Bi objetivo; referente al

experimento de la evolución de Harrold y su resultado, mejora la detección de fallas, no está ambientado a un ambiente industrial y los ejemplos son pequeños.

- ***Prueba de regresión por selección***

En esta categoría la estrategia se enfoca en detectar las partes modificadas de un programa bajo un análisis estático de caja blanca.

Formalmente, dado un programa P , la versión modificada de P , P' y un conjunto de casos de prueba T , el problema consiste en determinar un subconjunto de T , T' con el cual P' puede ser verificado.

A continuación, las técnicas que se encuentran en esta categoría.

- ***Basados en grafos, análisis de dependencia y UML***

Chhabi Ravi Panigrahi y Rajib Mall (2012) proponen una técnica de asociado en condiciones de códigos de programas con orientación a objetos y el modelo del diagrama UML y de la máquina de estados de las clases afectada por un cambio. El modelo del programa representa la información utilizando posteriormente las aplicaciones de acuerdo con elementos del modelo afectados, así como los elementos de la transición en el modelo de transición de evolución empírica, muestra una mejoría con su 27.89% en la evolución de fallas, mejora la detección de fallas, no está ambientado a un ambiente industrial y los ejemplos son productos de software pequeños.

Partha Pratim Ray (2011) propuso una técnica y metodología para la localización de fallas en programas orientados a objetos, usando estrategias

de UML como los diagramas de estado, además de la representación del programa mediante grafos dirigidos, que sirven para la detección de fallas utilizando ejecución simbólica, lamentablemente la propuesta está en fase de implementación.

Biswas, Mall y Satpathy (2009) presentan una combinación de técnicas que se basa en la representación del programa a través de técnicas clásicas como grafos, en la que se añadió la información característica de sistemas embebidos (ECIDG) como es manejo de flujo de control, tareas, prioridad de tareas, manejo de excepciones, así como la información de los objetos relacionados, se combina con algoritmo de particionamiento para obtener los casos de prueba que deben optimizarse con el algoritmo genético; los resultados son alentadores, la propuesta ha sido aplicada en aplicaciones de pequeña escala.

Thomas Ball (1998) presenta una variación de la estrategia RH para RTS (Rothermel y Harrold) usando en análisis del flujo de control y análisis de la cobertura de flujo, pues se considera que la precisión de RTS puede ser ofrecida por información de cobertura recogida, es decir, combina información estática de análisis de programa y lo combina con información dinámica como la cobertura de flujo; esta técnica no ha sido evaluada experimentalmente en la academia, los algoritmos no han sido extrapolados para manejar flujo de control interprocedural.

Rothermel y Harrold (1997) presentan técnicas basadas en la técnica de representación de programas por medio gráfico para el flujo de control para productos o programas procedurales y demostró que bajo ciertas condiciones estas técnicas son seguras, el componente experimental fue llevado a cabo por medio de la aplicación de dos herramientas denominadas Dejavu1 y Dejavu2 para sus algoritmos intra e interprocedural en ambientes académicos controlados; el tamaño de la suite de test depende de factores como la

estructura del programa, el tipo de modificaciones y el diseño de la suite de pruebas.

- *Basado en modelos*

Swarnendu Biswas *et al.* (2013) proponen un modelo para pruebas de regresión basado en grafos de modelos que capture los factores relevantes para sistemas embebidos, por ejemplo, tareas, concurrencia de tareas, temporizaciones de tareas que se lo ha llevado a través de comunicaciones y priorización de entre procesos, y que en algunos casos se implementan con la ayuda del razonamiento humano, ejemplo de ello son los aportes de Harrold (1988) y Biswas (2009); para la implementación se trabaja con un lenguaje particular para sistemas embebido denominado RTSEM bajo MisraC, en lo referente a la experimentación, se ha trabajado en experimentación de casos con programas embebidos de tamaño reducido.

Swarnendu Biswas *et al.* (2009) presentan una combinación de técnicas que se basa en la representación del programa a través de técnicas clásicas como grafos, en la que se añadió la información característica de sistemas embebidos (ECIDG) como es manejo de flujo de control, tareas, prioridad de tareas, manejo de excepciones, así como la información de los objetos relacionados, se combina con algoritmo de particionamiento para obtener los casos de prueba que deben optimizarse con el algoritmo genético; los resultados son alentadores, la propuesta ha sido aplicada en aplicaciones de pequeña escala.

- *Basado en heurísticas*

Siavash Mirarab, Akhlaghi y Tahvildari (2012) presentan el problema de RTS de tamaño limitado en donde los testadores de código necesitan

estudiar cuántos test cases se pueden ejecutar en un tiempo limitado, para ello se usa un enfoque multiobjetivo temporizado de problemas en uno de programación entera usando dos criterios diferentes tomados de la cobertura y usa selecciones limitadas para encontrar puntos soluciones con próximos a la idea, sin cambios para obtener una solución final del necesario de votación a casos de prueba seleccionada, ahora debe su programación mediante un algoritmo Greedy que maximice la cobertura mínima de un manera técnica; lo propuesto ha sido evaluado en laboratorios con aplicación que van desde las 80klog programas pequeños, no en entorno industrial, por lo tanto se puede esperar que el desempeño sea diferente, herramientas a nivel de prototipo.

- *Basado en datos históricos*

Wikstrand, Feldt y Gorantla (2009) presentan un método de determinar qué casos de pruebas deben re-ejecutarse en una prueba de regresión, este método se fundamenta en la información histórica de las modificaciones de código a nivel de archivo, por lo tanto, relaciona la información del número de veces que un archivo se ha modificado y lo trata como información de cobertura del TC relacionado al archivo, según esta ordena los TC, solo considera test fallidos, solo TC relacionados con archivos corregidos son considerados, y además considera la evaluación de las dos últimas versiones del programa no con las versiones donde esos casos de prueba fueron considerados.

Kim, Zimmermann, Whitehead y Zeller (2007) proponen un algoritmo (técnica) para la predicción de errores basado en la información histórica e errores y sus correcciones, bajo la hipótesis de que los errores a futuro aparecen bajo una estrecha relación entre ellos, es decir, existe una relación entre las localizaciones de las fallas y sus correcciones, relaciones que se generan en modificaciones, eliminaciones, e incremento de código, considera

factores de precisión y granularidad para dar un enfoque de calidad de software; dentro de su evaluación aplicada a 7 sistemas de mediano y gran tamaño se desprende que existe una mejora en la predicción de fallas que dice que con un 10% de código fuente se puede predecir en un rango de 73% a 95% de fallas. Las aplicaciones utilizadas no son representativas, y los resultados no se los puede extrapolar a las aplicaciones cerradas, la calidad de la predicción está en relación de la calidad de la data almacenada.

- *Basado en detección de fallos*

Partha Pratim Ray (2011) propuso una técnica y metodología para la localización de fallas en programas orientados a objetos, usando estrategias de UML como los diagramas de estado, además de la representación del programa mediante grafos dirigidos, que es utilizada para la detección de fallas utilizando ejecución simbólica, lamentablemente la propuesta está en fase de implementación.

Ficco, Pietrantuono y Russo (2011) proponen una técnica de detección de fallas mediante la comparación del “comportamiento del sistema” en cada ciclo de desarrollo, su técnica de localización de fallas está embebida en el proceso de TDD y hace uso de la técnica de análisis dinámico basado en huellas de ejecución del sistema que constituye el modelo comparado correcto del sistema básicamente en las interacciones de métodos. Después de que en el grupo de funcionalidades se han implementado un ciclo de pruebas de regresión, se ejecuta con las comparaciones de los campos tanto del sistema y se realiza de violaciones, siendo las fallas aquellas que están más cerca de la falla; el experimento se llevó a cabo con una aplicación de 200k log con 560 clases en un ambiente de laboratorio industrial, falla cuando la modificación está dentro de un método (granularidad fina).

- *Basado en modificaciones*

Soetens, Demeyer y Zaidman (2013) proponen un modelo y una técnica contextualizada en desarrollo guiado por pruebas (TDD) para detectar a nivel de método en el paradigma orientado a objetos o los casos de prueba que están implicados en esos cambios o modificaciones, generándose la selección de casos de prueba para la regresión; el modelo que representa los cambios considera las modificaciones, adiciones y la eliminación de elementos en un sistema, generándose las dependencias de cambios. El algoritmo de selección trabaja con las dependencias entre los elementos modificados; se utilizó varios plugins para Junit para la construcción de la herramienta denominada CHEOPS, la cual se aplica a la herramienta de integración continua Cruise Control; los casos de estudio han sido evaluados empíricamente usando aplicaciones de varios cientos de miles de códigos en un entorno industrial.

Muslu, Brun, Ernst y Notkin (2013) desarrollan Solstice, una herramienta tipo plugin para el IDE Eclipse de Java (TDD) que implementa el enfoque de replicación del código base del desarrollo con propósito de realizar un análisis continuo del código. Este enfoque fue ampliado a tres conceptos del desarrollo: 1) búsqueda continua de errores, 2) PMD continuo, y 3) testing continuo, este último presentado en pantalla del desarrollo los efectos de una prueba y sus implicaciones (análisis textos); pequeñas modificaciones pueden llevar un intensivo análisis extenso.

Steinert, Haupt, Krahn y Hirschfeld (2010) denominan su técnica como continuous selective testing (CST) y utilizan la representación interna del programa en la herramienta de desarrollo para seleccionar automáticamente los test que ejecutan un regresión, para ello emplea análisis dinámico y usa la comparación de programas para la determinación de la selección continua de

puntos; la función es que este enfoque reduce este desarrollo en Squeak Smalltalk, sensible a pequeños cambios en los métodos, representación binario de programas y análisis dinámico; no ordena el test seleccionado y no ha sido evaluado en entornos industriales.

Orso, Shi y Harrold (2004) presentan una técnica de dos fases para realizar RTS seguras y que es aplicable a sistemas de software desarrollados bajo lenguaje Java que van desde los 700 a 5000 LOC (pequeños programas), así también presentan su herramienta denominada Dejavu para la parte experimental. La primera fase analiza el programa e identifica jerarquías entre entidades, relaciones entre las clases e interfaces, trabaja con análisis sintáctico para contar los cambios, así como las relaciones entre las clases; la segunda fase realiza la selección basada en la información de la partición identificada en la fase 1; las conclusiones muestran que la técnica es eficiente y reduce el esfuerzo de regresión.

Duarle, Cirne, Brasileiro y Machado (2006) proponen un mecanismo de pruebas considerando una granja de computadoras (GRID) bajo el enfoque de Junit para proyectos de mediana y gran escala implementando la herramienta para evaluación de su arquitectura que supere los problemas de la heterogeneidad del código, así como mejorar los tiempos de ejecución de prueba en entornos de desarrollo Junit/TDD; no se ha probado el impacto de la Grid sobre las métricas de cobertura.

- *Prueba de regresión por priorización*

El objetivo de este enfoque es priorizar los casos de prueba de acuerdo con algún criterio para mejorar el desempeño de una prueba de regresión. El objetivo principal es encontrar una permutación de una secuencia de casos de prueba.

Formalmente, dado un conjunto de casos de prueba T , PT el conjunto de permutaciones de T , una función PT a números reales $f: PT \rightarrow \mathbb{R}$ encontrar T' perteneciente a PT donde $(\forall T'') (T'' \in PT) (T'' \neq T') [f(T') \geq f(T'')]$. A continuación, las técnicas que se encuentran en esta categoría.

- *Basado en detección de fallas*

Albert Pravin y Subramaniam Srinivasan (2013) proponen un algoritmo para reordenar los casos de prueba priorizándolos con base en su tiempo de detección de fallas (tiempo de falla). El algoritmo de detección de falla es utilizado en este caso, para la parte experimental fue aplicado a WebKit que es una aplicación de software libre. La experimentación es llevada a cabo mediante la experimentación de casos y los resultados no se prestan para la generalización, ya que está desarrollado en código abierto y con aplicaciones de mediana escala.

- *Enfoque probabilístico basado en datos históricos*

Anderson *et al.* (2014) proponen una técnica de selección de casos de prueba de tal manera que tengan la más alta probabilidad de detección de falla, para ello considera información almacenada en los repositorios de un ambiente de desarrollo, como por ejemplo código fuente, código de pruebas, datos de correcciones, resultados históricos de test anteriores, y luego aplicando un enfoque de análisis de minería de datos se pueda determinar los mejores test que puedan detectar fallas en artefactos más propensos a presentar fallas; como caso de estudio se aplicó a un programa de Microsoft denominado Microsoft Dynamics AX, aplicativo con alrededor 5 millones de

LOC. 65.000 casos de prueba han sido evaluados empíricamente usando aplicación de varios cientos de miles de códigos en entorno industrial.

- *Basado en modelos*

Chhabi Ravi Pavigrahi y Rajib Mall (2010) proponen una técnica de RTP para programas orientados a objetivos, así, construyen un modelo por medio de grafos para representar información del código, como por ejemplo la dependencia de datos y control relación estético de objetos; cuando una modificación se presenta, la modificación es identificada por medio de un componente de código, los cambios son marcados en el modelo y con él se analiza o determina qué elemento del modelo presente fue afectado por el cambio, lo mismo se hace con el código del test y el modelo para verificar qué elementos son indirectamente testeados por el caso de prueba, se aplica alta prioridad a los casos de prueba que cubren mayor número de elementos de modelo, luego finalmente se aplica el caso de prueba de acuerdo a sus propiedades; no considera información dinámica del programa y aumenta la detección de fallas en comparación con otros enfoques relacionados.

- *Basado en análisis dinámico y lenguaje natural*

Hagai Cibulski y Amiram Ye Hudai (2011) introducen TestRank, una herramienta que usa combinaciones de varias técnicas de priorización susceptibles al cambio trabaja con análisis dinámico, evaluación del lenguaje natural del código y de la suit de pruebas. Las pruebas experimentales muestran que TestRank acelera el ciclo de prueba TDD, trabaja con proyectos de mediana escala y en ambientes académicos, además, no posee la característica de generalidad, ya que solo ha sido valorado con dos proyectos.

- *Basados en grafos, análisis de dependencia y UML*

Arilo Claudio Díaz Neto *et al.* (2008) presentan una estrategia para pruebas de integración en sistemas orientados a objetos basados en esquemas de priorización de acuerdo al peso de las dependencias entre clases diferentes del modelo clásico top down o bottom up; el proceso de experimentación fue llevado a cabo aplicando muestra lógica de software en ambiente industrial y académico; se desarrolló la herramienta denominada FAROL como resultado de nuestra reducción de los esfuerzos del testing considerando la reducción del número de stub para las clases con dependencia y que no pueden ser testeados conjuntamente; es necesaria la automatización para enfrentar proyectos grandes.

- *Basado en heurísticas*

Charan *et al.* (2013) presentan una herramienta de apoyo para el grupo de testadores que permite reducir las suites de pruebas bajo un enfoque de Pareto. La herramienta denominada REGRESSAID (MQHDE) experimentalmente es comparada con otros enfoques de la línea Greedy NSGAII, mostrando que la propuesta es superior en relación con la eficacia. REGRESSAID implementa dos algoritmos de optimización multi objetivo: MQHDE propuesto anteriormente por Charan Kumari *et al.* (2013) y el segundo algoritmo NSGAII que sirven para tratar los problemas de optimización tipo Pareto; no se ha realizado evaluaciones en sistemas industriales de gran tamaño.

Camila Brito *et al.* (2010) proponen una metaheurística basada en algoritmos Greedy denominada REACTIVE GRASP para priorizar los casos de prueba de acuerdo con sus métricas de cobertura (de bloque, de sentencia, y de ramificación). Compara su eficiencia con otras opciones de la línea de ingeniería basada en búsquedas otros algoritmos Greedy y algoritmos

genéticos, mostrando que Reactive Grasp es superior de acuerdo al proceso experimental presentado estudiando cinco proyectos que van desde 370 - 9500 LOC, considerando los términos de cobertura y solucionando el problema marcado por Rothermel y Li, que indican que los algoritmos Greedy fallan al escoger el ordenamiento óptimo de los casos de prueba, resultados no muy estables cuando se lo compara con otras alternativas.

- ***Prueba de regresión por optimización***

En esta categoría se encuentran aquellas técnicas que toman como bases técnicas de inteligencia artificial, entre ellos aprendizaje de máquina o técnicas de optimización, a continuación, las técnicas en esta categoría.

- ***Basado en lógica difusa***

En el trabajo de Manoj Kumar *et al.* (2012), se presenta dos contribuciones: la primera es el reconocimiento del carácter de incertidumbre de la labor de las pruebas de software, y el segundo es un modelo de medición de casos de prueba basado en el criterio de fitness y adecuancia para la clasificación de los casos de prueba basados en técnicas de inteligencia artificial (lógica difusa); se considera que existen diversos factores en los casos de prueba que en algunos casos al momento de la evaluación son antagónicos o contradictorios, por lo que se hace de la ponderación de aquellos factores y se los optimiza mediante programación multi objetivo aplicando lógica difusa, lográndose así definir diversos criterios de adecuancia para la evaluación de los casos de prueba y lograr la optimización del modelo matemático de optimización multi objetivo.

- *Basado en heurísticas*

En el trabajo de Siavash Mirarab *et al.* (2012), se presenta el problema de RTS de tamaño limitado, en donde los testadores de código necesitan estudiar cuántos casos de pruebas se pueden ejecutar en un tiempo limitado, para ello se usa un enfoque multiobjetivo temporizado de problemas en uno de programación entera usando dos criterios diferentes tomados de la cobertura y usa selecciones limitadas para encontrar puntos soluciones con próximos a la idea sin cambios para obtener una solución final del necesario de votación a test case seleccionada, ahora debe su programación mediante un algoritmo Greedy que maximice la cobertura mínima de una manera técnica; lo propuesto ha sido evaluado en laboratorios con aplicación que van desde las 8 KLDC programas pequeños, no en entorno industrial, por lo tanto se puede esperar que el desempeño sea diferente, con herramientas a nivel de prototipo.

R. Owen Robers (2004) presenta una estrategia de testing bajo el enfoque del desarrollo ágil, específicamente de integración continua, para proyectos de gran escala sugiere desarrollar código estableciendo longitudes de máximo de aplicativos errores ejecutados, para ello establece pequeños equipos de desarrollo con servidores de interacción locales y de hermandades conjuntamente como una disciplina para ejecutar integración continua y así poder enfrentar proyectos de gran escala y estrategias de integración continua para proyectos grandes.

2.4.2. Ingeniería de software con acceso a datos

La ingeniería de software es una disciplina formada por un conjunto de métodos, herramientas y técnicas que se utilizan para el desarrollo de los programas informáticos (software). Esta disciplina trasciende la actividad de

programación, que es el pilar fundamental a la hora de crear un software (Pressman, 2010).

La ingeniería de software incluye actividades como el análisis, el diseño del proyecto, el desarrollo del software, las pruebas necesarias para confirmar su correcto funcionamiento y la implementación del software.

Desde el enfoque como proceso, la ingeniería de software comprende varias etapas como fases que se conoce como ciclo de vida del software, el cual está formado por cuatro etapas fundamentales: análisis, diseño, implementación y la implantación o liberación del producto software. Cada una con objetivos y resultados que sirven como entrada para la siguiente etapa, así, la etapa de análisis fija el alcance del proyecto y desarrolla el modelo de negocio; el diseño define el plan del proyecto, detalla las características y fundamenta la arquitectura; la implementación es el desarrollo del producto; y la implantación es la transferencia del producto software a los usuarios.

Un aspecto directamente relacionado con la ingeniería de software es la arquitectura de software, que consiste en determinar y esquematizar la estructura general del proyecto, diagramando su esqueleto con un grado relativamente alto de especificidad y señalando los distintos componentes que serán necesarios para llevar a cabo el desarrollo, tales como aplicaciones complementarias y diseño de bases de datos. (Vicente, E., Anticon, M. T., & Rivera, 2007)

Con relación a los datos y su enfoque en el diseño del producto, se considera tres enfoques: cómo se enfoca los datos desde la perspectiva del diseño estructurado de software, diseño orientado por objetos y diseño impulsado por modelos (Guzmán et al., 2017).

- *Acceso a datos en arquitectura estructurada de software*

Bajo esta perspectiva, al software se le considera como un conjunto de módulos y sus relaciones, luego del proceso de análisis, se realiza una descomposición en modular en jerarquías que permitan la implementación sin involucrar elevados costos de mantenimiento; la idea original del diseño estructurado fue presentada en la década de los 70, por Larry Constantine en Amo, Normand, y Pérez (2005), y continuada posteriormente por otros autores: Myers, Yourdon y Stevens (Brooking, 1984).

La característica básica del módulo es que es vista como una caja negra con interfaces de datos de entrada y salida y una lógica de procesamiento estructurado realizado por el módulo. Desde el punto de vista de sistemas de software distribuidos, la arquitectura de construcción del producto software va por la orientación del diseño por capas, la arquitectura puede ser de dos y tres capas, dependiendo la estrategia de minimización de la complejidad.

Dos capas (cliente – servidor): En donde se considera que los módulos que van al lado del cliente son los que consumen servicios y los módulos que van al lado de servidor son los que proveen servicios de datos.

Tres capas (presentación, negocio, datos): En esta estrategia se pretende una mayor separación de funcionalidades, principalmente para el acceso a los datos, que son trabajados por tecnologías como CORBA, ODBC y JDBC (Gorton, 2006).

- *Acceso a datos en arquitectura de software orientado a objetos*

Es un enfoque de la ingeniería de software que modela un sistema como un grupo de objetos que interactúan entre sí modelando el mundo real. De manera similar, luego del proceso de análisis, se procede a modelar el mundo real utilizando técnicas y herramientas como UML (casos de uso, diagramas de clases, diagramas de secuencia, diagramas de estado) (Dennis et al., 2015).

Desde el punto de vista de la arquitectura de construcción del software, básicamente la estrategia utilizada es la de sistemas en tres o más capas, dependiendo de las necesidades, la estrategia es dividir en capas la complejidad del software (Garlan, 2014).

Bajo este enfoque, el acceso a datos se trabaja básicamente con una capa de datos o de persistencia de datos. También existe una capa que trabaja la lógica del software conocida como capa de negocio, aquí se procesa por peticiones del usuario, finalmente, la capa de presentación o de interfaz es la capa percibida por el usuario para la visualización de los datos y la información.

La tecnología para el acceso a los datos que actualmente se usa es ODBC y JDBC, implementado por el lenguaje de programación y el sistema manejado de bases de datos.

- *Acceso a datos en diseño de software dirigido por modelos*

La ingeniería de software impulsada por modelos es la explotación de las ideas de ingeniería de modelos al desarrollo del proceso software (MDE por sus siglas en inglés), en este paradigma contemporáneo se encuentra el

desarrollo impulsado por modelos (MDD por sus siglas en inglés) y también la arquitectura impulsada por modelos (MDA por sus siglas en inglés) propuestas por el Object Management Group (OMG). MDE define los mecanismos para utilizar modelos en la automatización de tareas propias del desarrollo de software, por ejemplo, configuración, análisis y diseño, generación de código, refinamiento, refactoría, traducción a otros lenguajes o plataformas, entre otras. Los planteamientos de MDE hacen hincapié en aspectos como la construcción de modelos en exhaustiva comunicación con los usuarios, para que tengan sentido desde su punto de vista; adicionalmente, sugiere que dichos modelos sirven como base para poner sistemas en ejecución. Básicamente se continúa utilizando UML como lenguaje de modelado y DSL como lenguaje de definición de perfiles de dominio. (Jon Whittle, John Hutchinson, Mark Rouncefield, 2014)

Por otro lado, MDA propone la definición y uso de modelos en el proceso de desarrollo de software a diferente nivel de abstracción, así como la posibilidad de la generación automática de código a partir de los modelos definidos y de las reglas de transformación entre dichos modelos (Garlan, 2014).

Los modelos a los que se hace referencia son:

- CIM: Es el modelo independiente de la computación, surge en la fase inicial del proceso de desarrollo comprendiendo la modelación del negocio en su totalidad.
- PIM: Representa los modelos que describen una solución de software que no contiene detalles de la plataforma concreta donde será implementada la solución, de ahí su nombre de modelos independientes de la plataforma. Estos modelos surgen como resultado del análisis y diseño.

- PSM: Los modelos específicos de la plataforma. Surgen del PIM y se crean entre las fases del diseño y la codificación de la solución. Luego, el código se genera después de la codificación y las pruebas.

El proceso de desarrollo de software basado en MDA presupone un cambio en la secuencia de actividades para la obtención de un producto informático. A pesar de que el ciclo de vida del producto informático mantiene sus etapas desde la captura de requisitos hasta el despliegue de la aplicación; el uso de este marco de trabajo deviene la adaptación del ciclo de desarrollo de software. Una vez que se avanza en las etapas dentro del ciclo de desarrollo de software, se va centrando el trabajo en la construcción de los modelos, su refinamiento y la posterior transformación. En ese sentido, el uso de modelos y estándares que garanticen la independencia de las diversas plataformas, abiertas o propietarias, como servicios web, .Net, Corba, J2EE, u otras es ampliamente utilizado, así mismo, la estrategia de desarrollo por n capas.

2.4.3. Técnicas de pruebas de regresión en software con acceso a datos

En el contexto de productos de software con acceso a datos, las técnicas de regresión se siguen enfocando en los conceptos de minimización, priorización, selección y optimización, pero considerando los aspectos de la bases de datos y los datos almacenados combinados con aspectos de caja negra, firewall, representación de grafos cuando de código se trata, ese el caso de Haraty et al. (2001), quienes consideran las pruebas de regresión desde la perspectiva de procedimientos almacenados con un enfoque de cortafuegos. Willmor y Embury (2005) presentan su técnica que utiliza el enfoque de caja blanca con la representación de grafos para el código. Con respecto a la consideración del lado de la base de datos y su lenguaje de

consulta, Suarez-Cabal y Tuya (2005) proponen una métrica de cobertura para consultas SQL como criterio de suficiencia en este tipo de productos software. En ese mismo contexto, Tuya et al. (2008) realizan una serie de experimentos para evaluar comparativamente las diferentes técnicas desde el enfoque de cobertura de las instrucciones SQL (Structured Language statements), con el objetivo de seleccionar los casos de prueba que están relacionados con la base de datos. Más tarde, Erik Rogstad et al., (2013) consideran el aspecto transaccional de los sistemas bajo el enfoque de caja negra usando técnicas de clasificación para determinar la similaridad de los casos de prueba. Ya en el contexto de la técnica de clusterización, E. Rogstad y Briand (2015) consideran los casos de prueba relacionados a la base de datos en grupos bajo el concepto de desviación (diferencias entre las manipulaciones de la base de datos en diferentes versiones del sistema), para ello utiliza medidas de entropía de información propuestos por Guo, Subramaniam y Chundi (2012) para evaluar las diferentes técnicas de clusterización. En esa línea hay propuestas que se analizan en el campo de la inteligencia artificial, precisamente en la minería de datos, donde J. Zhang et al. (2017) proponen un trabajo implementando un modelo de clusterización supervisada en conjunción con lógica fuzzy; así mismo, en la línea de clusterización, se encuentra trabajos como el de Vidyanand, C., y Ramesh, G (2017), quienes proponen una técnica que se basa en las desviaciones de los datos entre diferentes instantes de una bases de datos. En el campo de la optimización existen estudios que propone heurísticas Greddy, como es el caso de Shi, Thummalapenta, Lahiri, Bjorner y Czerwonka (2017), quienes trabajan bajo perspectiva de considerar al producto software como una colección de casos de prueba relacionados de la misma manera que los módulos que componen el producto software, trabajo que se ocupa de casos de estudio de escala grande. D. Marijan, M. Liaaen, A. Gotlieb, S. Sen and C. leva (2017) proponen una tecnología TITAN bajo el enfoque de desarrollo iterativo incremental (integración continua) con métricas de tiempos para estimar costo beneficio de las pruebas bajo el enfoque combinado de minimización y priorización.

2.4.4. Técnicas de clusterización para pruebas de regresión en software con acceso a datos

En el enfoque de analizar los casos de pruebas, se observa que algunos casos de pruebas tienen comportamientos similares y que en ocasiones provienen de la misma falla, lo que ha motivado investigaciones para analizar los casos de pruebas en grupos basados en métricas determinadas (Ammann & Knight, 1988).

Se ha usado metadata de software o especificaciones de funcionalidades. Vangala *et al.* (Vangala et al., 2009) usaron perfiles de ejecución y ejecuciones estáticas para comparar casos de pruebas y aplicar algoritmos de clusterización. Orso *et al.* (2004) usaron metadata en formato XML para seleccionar los casos de pruebas. Zimmermann *et al.* (2007) registraron una técnica que combina el histórico de ejecuciones con las modificaciones de código. Wikstrand *et al.* (2009) presentaron una técnica que relaciona los resultados de ejecuciones de las pruebas con los defectos encontrados y los históricos de modificaciones. Ficco *et al.*, (2011) presentaron una técnica basada en la capacidad de detección de fallos de los casos de pruebas bajo la metodología de desarrollo iterativo incremental.

Vangala et al. (2009) considera que existe conexiones entre los grupos de casos de pruebas que en ocasiones están ocultas, siendo necesario determinar patrones entre ellos. En las últimas décadas se ha venido trabajando en la aplicación de técnicas de softcomputing como las propuesta por Gökçe et al., (2015), en minería de datos por Anderson *et al.* (2014), redes de colmenas por Maia, do Carmo, de Freitas, Gomes, de Campos, de Souza, et al. (2010) y aprendizaje de máquina por Engström et al. (2010). En esta última, las técnicas de análisis de clúster se han considerado estudiar el clúster de casos de pruebas.

Dickinson *et al.* (2001) estudiaron minería de datos para filtrar y clusterizar los casos de pruebas basados en sus perfiles de ejecución y perfiles de programa, demostrando que la técnica de análisis de clúster puede apoyar con confianza las pruebas de regresión de software (A. Podgurski *et al.*, 2003; Shin Yoo *et al.*, 2009; Hall *et al.*, 2009; Cartaxo *et al.*, 2011).

En el contexto de software con acceso a datos, Haraty *et al.* (2001) consideraron las pruebas de regresión desde la perspectiva de procedimientos almacenados con enfoque de firewall. Willmor y Embury, en el 2005) presentaron su técnica de prueba de regresión segura usando un enfoque de caja blanca según la representación de grafos para el código de la aplicación. Tuya *et al.* (2009) propusieron una serie de experimentos para evaluar y comparar la eficiencia de varias técnicas de caja blanca para instrucciones SQL (Structured Query Language) con el objetivo de seleccionar casos de prueba asociados a bases de datos. Rogstad *et al.* (2013) consideraron los aspectos transaccionales de las bases de datos, pero desde la perspectiva de caja negra, haciendo uso de árboles de clasificación para representar similitudes de los casos de prueba.

De manera similar, Rogstad and Briand (2015) consideraron casos de prueba relacionados a bases de datos tratando las desviaciones o diferencias entre distintos momentos de una base de datos, de acuerdo con las diferentes manipulaciones de los datos y según las versiones de los programas; este enfoque está basado en criterios de entropía para evaluar diferentes de clusterización. La Tabla 6 muestra un resumen de las características de estos trabajos.

Tabla 6. Resumen de técnicas de clusterización para software con acceso a datos

Autor(es)	Técnica de regresión	Feature Analizada	Técnica de clusterización usada	Enfoque de clusterización	Contexto del producto software
Haraty <i>et al.</i> (2001)	Minimización	Estructuras de sentencias SQL	Graph walk, firewall	No aplica	Académico
Willmor & Embury (2005)	Selección	Código asociado a las bases de datos	Control flow graphs	No aplica	Académico
Rogstad <i>et al.</i> (2011)	Selección	Grafos de control de flujo a nivel procedural	k-means	Supervisada	Académico
Rogstad <i>et al.</i> (2013)	Selección	Estructuras UML de los casos de prueba	DART Model	No supervisado	Industrial
Rogstad & Briand (2015)	Selección	Patrones de desviación en casos de prueba en manipulaciones de bases de datos, priorización de requerimientos	Expectation maximization	No supervisada	Industrial

CAPÍTULO III: METODOLOGÍA DE INVESTIGACIÓN

Una parte importante de la investigación es la decisión de la metodología de investigación apropiada, en este caso se aplicará el método científico aplicado a la ingeniería de software denominada ingeniería de software empírica, que tiene como objetivo producir conocimiento confiable y aplicable al desarrollo del proceso del software. El enfoque empírico permite obtener evidencia para probar las hipótesis sobre técnicas y herramientas para construir software, obteniendo así conocimiento confiable para fundamentar las afirmaciones con respecto a las pruebas de regresión de software. (Matalonga y Solari, 2013)

La ingeniería de software empírica aplica el método científico experimental para la construcción del conocimiento, propuestas de Wohlin et al. en (2012) y Barbara Kitchenham et al. en el (2009), para ello realiza una observación rigurosa y la manipulación de la realidad.

Los métodos de investigación empíricos van desde las encuestas y casos de estudio hasta estudios correlacionales y experimentos controlados (N. Juristo et al., 2006).

En esta investigación se aplicará el método de investigación experimental denominado caso de estudio para recolectar información sobre el método de regresión de pruebas de software planteado. Se trata de un caso

de estudio con un enfoque explicativo, cualitativo flexible. Un caso de estudio involucra varios pasos: definición, diseño o planificación, ejecución del caso de estudio, análisis de datos y desarrollo de conclusiones (B. Kitchenham et al., 1995; Gómez et al., 2014).

3.1. Hipótesis

3.1.1. Hipótesis General

HG: El método de clusterización probabilística mejora las pruebas de regresión de software con acceso a datos en entornos de desarrollo iterativo incremental.

3.1.2. Hipótesis Específicas

- H1: El método de clusterización probabilística reduce la cantidad de casos de prueba en las pruebas de regresión de software con acceso a datos en entornos de desarrollo iterativo incremental.
- H2: El método de clusterización probabilística mejora la precisión de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental.
- H3: El método de clusterización probabilística mejora el rendimiento cambiando de las pruebas de regresión bajo software con acceso a datos en entornos de desarrollo iterativo incremental.

3.2. Identificación de Variables

3.2.1. *Variable Independiente*

VI: Método de clusterización probabilística.

3.2.2. *Variable Dependiente*

VD: Pruebas de regresión de software con acceso a datos en entornos de desarrollo iterativo incremental.

3.3. Operacionalización de variables

Tabla 7. Operacionalización de variables

Variable	Tipo de variables	Definición conceptual	Indicadores	Descripción del indicador
Método clusterización probabilístico	Variable independiente	El proceso de división de los datos en grupos de objetos similares, basados en la probabilidad de pertenencia al grupo(s)	Presencia - ausencia	Aplicación del método de clusterización probabilístico
pruebas de regresión de software con acceso a datos en entornos de desarrollo iterativo incremental	Variable dependiente	Ejecución de casos de pruebas que ya se realizaron, a fin de asegurar que las modificaciones y/o incrementos de funcionalidades no propagaron efectos colaterales no deseados	Cantidad de casos de prueba reducidos	Cantidad de casos de prueba filtrados que serán considerados a una prueba de regresión.
			Precisión	% Porcentaje de todos los casos de prueba relevantes filtrados respecto de los casos de prueba filtrados
			Rendimiento combinado	Cantidad, mide de la precisión global de la prueba. Se calcula a partir de la precisión y la recuperación de los casos de prueba

3.4. Matriz de Consistencia

Tabla 8. Matriz de Consistencia

Problema	Objetivo	Hipótesis	Variables	Metodología
Problema General	Objetivos General	Hipótesis General		
¿De qué manera el método de clusterización probabilística mejora de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental?	Generar un método de clusterización probabilística que mejore las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental	El método de clusterización probabilística mejora las pruebas de regresión de software con acceso a datos en entornos de desarrollo iterativo incremental.	Independiente: Método de clusterización probabilística Dependiente: pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental	Tipo de investigación Aplicada Diseño de la investigación Experimental puro Población Pruebas de regresión de cinco productos de software de mediano tamaño con accesos a datos desarrollados en entornos interactivo incremental Muestra La selección de la muestra se realizó en función a la factibilidad en el acceso a la información, respaldo institucional en la aplicación del método y los técnicos de desarrollo de los productos de software.
Problemas Específicos	Objetivos específicos	Hipótesis específicas		
¿De qué manera el método de clusterización probabilística reduce la cantidad de casos de prueba en las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental?	Reducir la cantidad de casos de prueba en las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental	H1. El método de clusterización probabilística reduce la cantidad de casos de prueba en las pruebas de regresión de software con acceso a datos en entornos de desarrollo iterativo incremental.		
¿En qué manera el método de clusterización probabilística mejora la Precisión de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental?	Mejorar la precisión de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental	H2. El método de clusterización probabilística mejora la precisión de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental.		
¿En qué manera el método de clusterización mejora el rendimiento cambiando de las pruebas regresión de software con acceso a datos en entornos de desarrollo iterativo incremental?	Mejorar el rendimiento cambiando de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental	H3. El método de clusterización probabilística mejora el rendimiento cambiando de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental.		

3.5. Metodología

Para la consecución de los resultados esperados, en este proyecto investigativo se ha seguido los siguientes pasos metodológicos, diferenciados cada uno de ellos: revisión de la bibliografía y establecimiento de marcos conceptuales, definiciones relacionadas con las preguntas de investigación establecidas en esta investigación, análisis de las técnicas de pruebas de regresión establecido en el marco referencial obtenido en anterioridad y, finalmente, el diseño de un método de pruebas de regresión de software de acceso a datos con su validación correspondiente a través del método experimental.

3.5.1. Tipo y diseño de la investigación

El tipo de investigación es cuantitativo y de diseño experimental de corte transversal. El diseño experimental de la prueba se aplica a dos grupos: grupo experimental y grupo de control, con las mediciones que corresponden a la aplicación del método propuesto al grupo experimental y la aplicación del método convencional al grupo de control, el efecto del tratamiento se obtiene considerando la diferencia entre las dos mediciones.

El objetivo del diseño experimental es aplicar la relación causa-efecto entre dos o más variables, en este tipo de diseño el investigador intencionalmente modifica el estado de algunos sujetos de estudio y manipula deliberadamente un tratamiento que desea estudiar o evaluar en el marco de una situación controlada (Lerma, 2009).

Por lo anterior, en este trabajo se observará la relación causa-efecto entre la variable independiente VI: “Método de clusterización probabilística” y

la variable dependiente VD: “Las pruebas de regresión de software con acceso a datos bajo entornos de desarrollo iterativo incremental”.

Es de corte transversal porque el estudio se realiza en un momento determinado y no interesa la evolución de las variables en el estudio mencionado.

3.5.2. Unidades de análisis

Las unidades de análisis de la investigación son las pruebas de regresión de las versiones (incrementos) de proyectos de software con acceso a datos bajo entornos de desarrollo iterativo incremental. Proyectos y versiones a los cuales se les aplicó los métodos de pruebas de regresión propuesto y el de método convencional (retestALL), realizándose las mediciones correspondientes, pudiendo así obtener los datos de las variables e indicadores sobre los cuales se realizaron los procesamientos y análisis estadísticos para obtener la información y, por consiguiente, las respuestas a las hipótesis planteadas.

3.5.3. Población de estudio

Definida las unidades de estudio, la población considerada son 32 pruebas de regresión de las 37 versiones correspondientes a cinco proyectos de software con acceso a datos de mediana y gran escala, en el ámbito administrativo académico, CRM y ERP, que actualmente se encuentran en producción en la Escuela Superior Politécnica de Chimborazo (ESPOCH), institución educativa que forma parte del Sistema Nacional de Educación Superior de la República del Ecuador (<http://www.educacionsuperior.gob.ec>), ubicada en la provincia de Chimborazo, cantón Riobamba y en el sector privado del país.

3.5.4. *Tamaño de la muestra*

La medición se aplicó a 32 pruebas de regresión de la población, considerando que a las primeras versiones no se les aplica una prueba de regresión.

3.5.5. *Técnica de recolección de datos*

Para la recolección de datos se usaron las siguientes técnicas e instrumentos en cada una de las etapas:

- Técnicas. - Análisis documental: Técnica que permite registrar o acumular datos en tablas o cuadros de información, siendo fuente de información creciente, que en este caso ha permitido registrar datos de casos de prueba de cada proyecto.
- Instrumentos: Los datos se recopilados en hojas de datos que registran datos, el número de casos de prueba de cada proyecto, el número de casos de prueba por cada funcionalidad en un ciclo de desarrollo, el número de casos de prueba luego de aplicar el método de prueba de regresión, así como los cálculos respectivos a cada uno de los indicadores considerados en esta investigación.

En la Tabla 9 se muestran los instrumentos de medición que serán utilizados para registrar las mediciones de las métricas y los indicadores considerados.

Tabla 9. Instrumentos para registro de métricas

indicador	Unidad de medida	Técnica	Instrumento
Cantidad de casos pruebas reducidos	Número	Análisis documental	Hoja de datos
Precisión	Porcentaje	Análisis documental	Hoja de datos
Rendimiento combinado	Número	Análisis documental	Hoja de datos

3.5.6. Métodos y técnicas de procesamiento y análisis de datos

Dado que las variables se formularon en valores numéricos, por consiguiente, se realizó un análisis cuantitativo. Para ello los datos se colocan de forma de una matriz de tabulación para procesarlos mediante el software estadístico R, MINITAB 18 y para registrar la hoja de cálculo Microsoft Excel, para el correspondiente análisis y obtener la información procesada.

En los datos obtenidos, cada columna representa los indicadores antes y después de aplicar la prueba de regresión al ciclo de desarrollo; cada fila representa un sujeto específico de la muestra utilizada.

Para analizar los datos colocados en la matriz de tabulación, se utilizará el análisis cuantitativo, que consiste en comparar diferencias entre grupos de la muestra (grupo experimental y grupo de control).

Para la prueba de las hipótesis se utilizará la prueba estadística T de Student con muestras independiente en un contexto experimental, donde se comparan las medias del grupo en dos técnicas diferentes. Las medias se calculan aplicando la fórmula estadística más conocida como la fórmula del promedio.

CAPÍTULO IV: EL APORTE

4.1. Modelo conceptual del aporte

El método de regresión de software con accesos a datos bajo el entorno de desarrollo iterativo incremental se fundamenta en dos características principales, la primera que el producto software debe contar con acceso a bases de datos del tipo relacional, el segundo que debe estar desarrollado bajo el paradigma orientado a objetos con la utilización de frameworks de desarrollo de pruebas unitarias como por ejemplo JUnit del lenguaje de programación Java.

La Figura 1 muestra una abstracción conceptual del método de clusterización probabilístico incremental que puede ser aplicado en cada iteración/versión en la línea del desarrollo del producto, para determinar los casos de prueba con accesos a datos que serán considerados en una prueba de regresión para determinar los fallos de regresión en aquella iteración/versión.

Para la obtención del aporte se siguieron los siguientes pasos:

- Recopilación y análisis los métodos para pruebas de regresión para software con acceso a datos.
- Diseño de una técnica de prueba de regresión para software con acceso a datos.
- Validación de la técnica de prueba de regresión por medio de la aplicación de un proceso de experimentación enfocado a la ingeniería de software, a través del método de un caso estudio embebido.

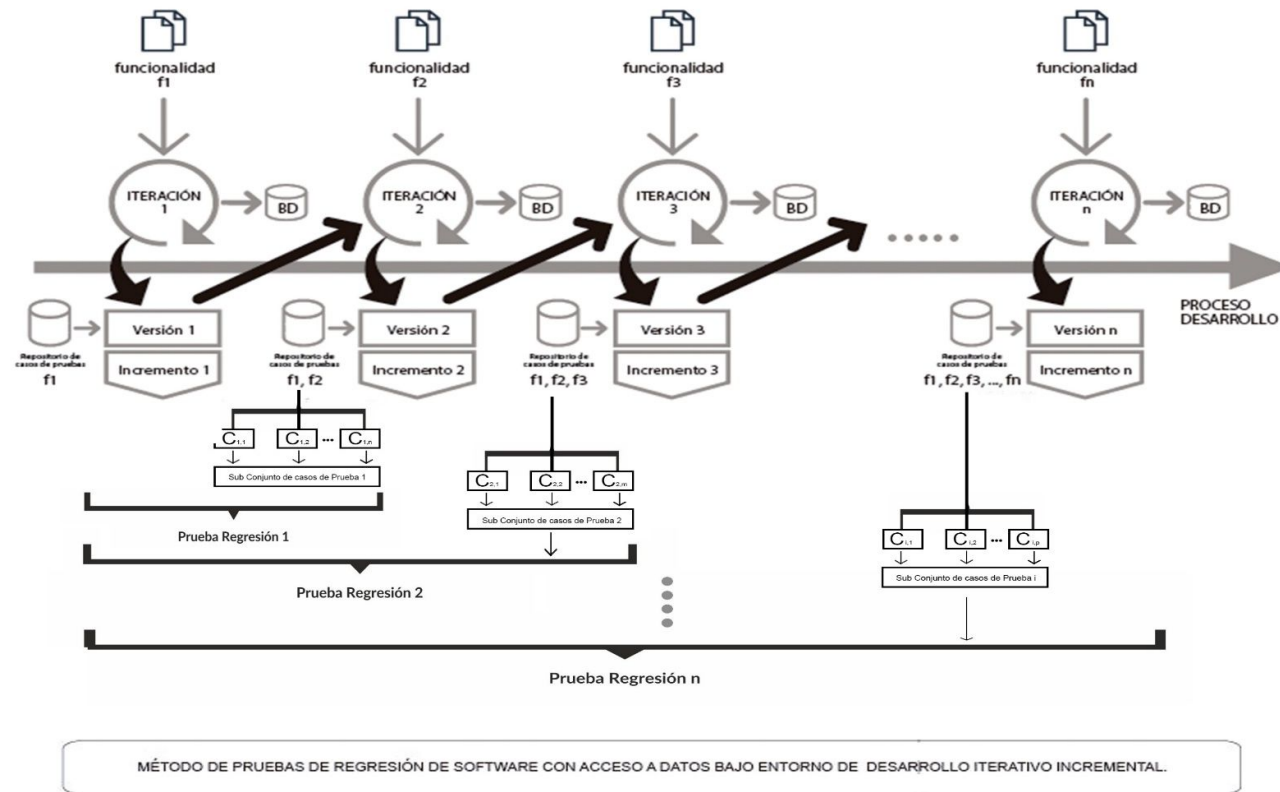


Figura 1. Diagrama conceptual del aporte
Fuente: Propia

4.2. Determinación de requerimientos no satisfechos

Después de analizar los trabajos relacionados a la investigación, se ha encontrado que existen trabajos que proponen técnicas de regresión desde enfoques de caja blanca y caja negra para aplicaciones de bases de datos. Respecto a enfoques que usen en enfoque de caja blanca, no se han encontrado trabajos que consideren, bajo un enfoque integral, las modificaciones al código fuente en conjunción con los accesos a la base de datos y sus casos de prueba relacionados.

4.3. Método para pruebas de regresión en software con acceso a datos

En esta sección se presenta el enfoque para agrupar casos de prueba para productos software con acceso a bases de datos relacionales para la realización de pruebas de regresión de software combinado, así como la técnica de pruebas unitarias de caja blanca con el esquema de una base de datos relacional en un entorno de desarrollo iterativo incremental.

4.3.1. El enfoque

La estrategia medular es agrupar los casos de prueba que estén relacionados con las interacciones sobre campos de las tablas de una base de datos, para luego, en una prueba de regresión, ejecutar el o los grupos de casos de prueba que contengan los casos de prueba que detectaron fallos.

A continuación, se describe los pasos del enfoque:

1. Selección de casos de prueba

El objetivo de este paso inicial es analizar los casos de prueba que tienen acceso o que realizan operaciones en los campos de las tablas de una BD (T), para ello se combina el análisis del código del producto software, tanto como el esquema de la BD, como resultado se obtienen los casos de prueba que son agregados al conjunto de casos de prueba (T) que acceden a la BD.

2. Construcción de la Matriz de similitudes de testDB

En este paso, la información referente a la tabla(s), y a sus columnas(s) de cada caso de prueba que interactúa con la BD se registra por medio de una matriz (M) de 0 y 1, donde un 1 representa interacción del caso de prueba con el atributo de la entidad/relación, como muestra la Tabla 10, generándose una matriz $M_{m \times n}$, en donde las m filas son los casos de prueba y las (n) columnas son los campos de las tablas de la base de datos del producto software.

Tabla 10. Matriz casos de prueba con acceso a datos

		Table				Table 1								Table n			
		Test case				Col1				Col2							
TC 1	1		0	0	1	...	0		0	1		1					
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
TC n	0		1	1	1		1		1	0		0					

3. Creación de grupos de casos de prueba con acceso a BD

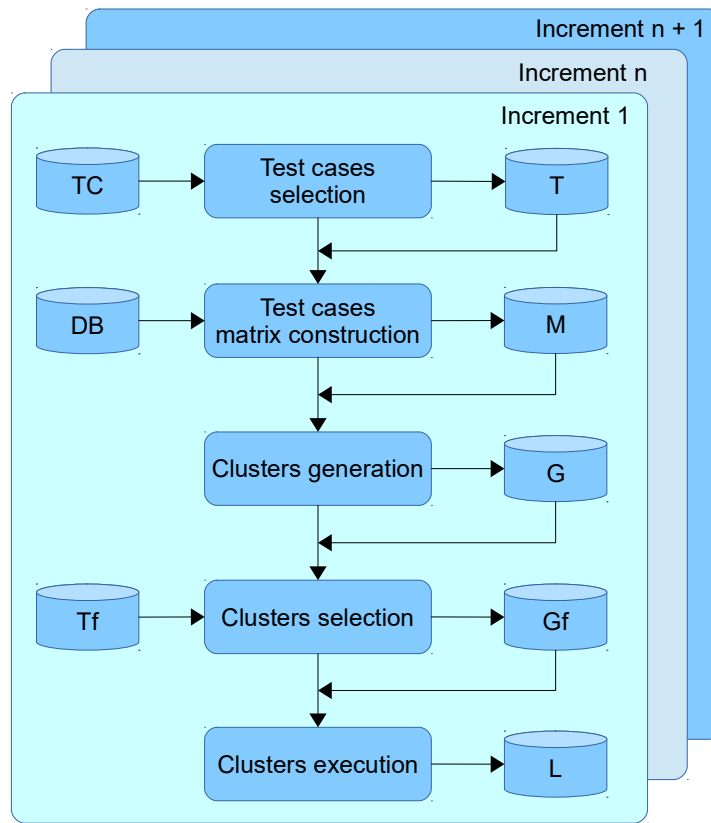
En esta etapa se hace uso del algoritmo de clusterización no supervisado denominado Expectation Maximization (EM), que permite analizar características probabilísticas de los casos de prueba (media y desviación estándar) y, con base en ellos, determinar su función de distribución probabilística para asignar la pertenencia del caso de prueba a un clúster basado en la matriz de similitud de la fase anterior. El algoritmo puede decidir cuántos clústeres crear basado en validación cruzada de los datos y utiliza el modelo Gaussiano finito de mezclas que asume que todos los atributos son variables aleatorias independientes. (Witten & Frank, 2005)

4. Determinación de grupos de casos de prueba

En este paso, cuando se añaden nuevas funcionalidades o se modifica alguna anterior, es decir, nuevos casos de prueba o modificación de los ya existentes, se agrupan los antiguos casos de prueba con los nuevos y se selecciona el o los grupos donde se ubicaron los nuevos casos de prueba, que en caso de falla deben ejecutarse todo el grupo de casos de prueba.

5. Ejecución prueba de regresión

En este paso, una vez determinados el clúster donde se ubicaron los casos de prueba con fallos, se procede a la prueba de regresión ejecutando todos los casos de prueba del o el grupo(s), generándose una lista de fallos relacionados, si es aplicable.



Terminology

TC:	Test cases of a given increment
T:	Set of test cases with access to the database
DB:	Database schema
M:	Matrix of similarities
G:	Clusters of test cases
Tf:	Set of test cases with faults
Gf:	Clusters of test cases with faults
L:	List of faults of the regression testing

Figura 2. Método para pruebas de regresión en software con acceso a datos
Fuente: Propia

4.3.2. Algoritmos diseñados para el método

A continuación, se describe los algoritmos que permiten implementar el enfoque, los cuales incluyen los pasos necesarios para desarrollar el método:

1. Determinación de casos de prueba que acceden a la BD
2. Construcción incremental de la matriz de similitudes de casos de prueba
3. El proceso de clusterización
4. La determinación de los clústeres con fallos
5. La ejecución de la prueba de regresión y la determinación de fallas de regresión

La Tabla 11 muestra la lista de identificadores usados en cada uno de los algoritmos.

Tabla 11. Identificadores usados en los algoritmos

Variable	Significado
BD	Data Base
T	Set of test case
F	Functionalities' Set
f_i	Functionalities i-esima $f_i \in F$
TC_i	Set of test cases from functionality i-esima $f_i \in F$
tc_i	Test case i-esimo of functionality
$tc_{i,j}$	Test case i- esimo $tc_i \in TC_i$ from funionality j-esima
TestBD	Set of test cases with access to BD
TestBDfailed	Set of test cases failed with access to BD
MatTestBD	Matriz of test case with access BD
MatClusterTestBD	Matriz of Clúster test cases with access DB
MatClusterRegressionTestDB	Matriz of Clúster test cases failed with access to DB
ListFailures	Set of failures for regression Test
NumTestDB	Number of test cases with access to DB

La lógica principal es realizar iterativamente pequeñas pruebas de regresión con cada una de las versiones del producto que por incremento o modificación de funcionalidades f_i se vayan construyendo, el algoritmo N.º 1 presenta los pasos básicos para ello.

1. Algoritmo principal

Por cada funcionalidad f_i , se realiza una determinación de casos de prueba que acceden a la base de datos, se aplica un proceso de construcción de matriz de similitudes basado en el accionar sobre los campos de la base de datos, una vez con la matriz se aplica un proceso de clusterización y finalmente se da la ejecución de los clústeres que contengan casos de prueba que hayan fallado y se muestra su listado de fallas.

Algoritmo N.º 1: Cluster Selection for Regression Test

Input: f_i , TC_i

Output: Failed's List

```

1 Process Regression
2   repeat
3     for each  $f_i$  do //  $f_i$  owner F
4       TestDB <- selectTestDB( $f_i$ ,  $TC_i$ )
5       MatTestBD <- buildMatrizTestDB(testDB)
6       MatClusterTestDB <- clusterizationEM(MatTestBD)
7       MatClusterRegressionTestDB <- (MatClusterTestDB  $\cap$  TestDBfailed)
8       executeClusterRegressionTestDB(MatClusterRegressionTestDB)
9     end for
10  until  $\nexists f_i$ 
11 EndProcess

```

2. Selección de casos de prueba

Para la implementación de esta fase del enfoque se desarrolló una herramienta software que realiza el análisis del esquema de la base de datos para determinar las operaciones de base de datos que son invocadas desde código del caso de prueba del

producto software, la salida de esta fase constituye los casos de prueba que acceden a la BD TestDB, como lo muestra el algoritmo N° 2: selectedTestDB.

Algoritmo N.º 2: selectedTestDB

Input: f_i , TC_i

Output: TestDB, NumTestDB

```

1 Process ChoseTestDB
2   numTestDB <- 0
3   repeat
4     if  $TC_i$  access DB then
5       TestDB <- TestDB U  $TC_i$ 
6       NumTestDB++
7     end if
8   until  $\nexists TC_i$ 
9 EndProcess

```

3. Construcción de la matriz de similaridad de casos de prueba

En esta fase, como presenta el algoritmo N.º 3: buildMatrixTestDB, se construye iterativamente la matriz MatTestDB de similitudes con n filas por cada caso de prueba que accede a la BD, m columnas por todos los atributos de cada entidad/relación. Así, en cada fila se representa con un 1 en caso de que el caso de prueba esté relacionado con algún atributo, y 0 en caso contrario. La herramienta desarrollada también es utilizada para esta fase.

 Algoritmo N.º 3: buildMatrixTestDB

Input: TestDB, numTestDB

 Output: MatTestDB

```

1 Process createMatrizTestDB
2   i<-1
3   while i < numTestBD do
4     MatTestDB <- ( MatTestDB ∪ (TestDBi))
5   end while
6 EndProcess

```

4. Proceso de clusterización

Para el procedimiento de clusterización, se consideró por razones prácticas la implementación de la herramienta de minería de datos Weka (por su documentación, activo foro de discusión y la existencia de APIS para código en JAVA) que implementa diferentes algoritmos de clusterización, de entre ellos se seleccionó el algoritmo probabilístico Expectation Maximization (EM), que tiene la ventaja de determinar el número k de clústeres con base en la información de los casos de prueba de la matriz de similitudes. (Jung et al., 2014)

 Algoritmo N.º 4: clusterizationEM

Input: MatTestDB

 Output: MatClusterTestDB

```

1 Process ClusterizationEM
2    $\Theta$  Vector desconocido de parámetros
3    $\Theta^0, \Theta^1, \dots, \Theta^T$ , T criterio de convergencia
4   T<-0
5    $\Theta^0$ <- 0
6   Repetir
7      $Q(\Theta, \Theta^t)$ <-  $E[\log p(x^g, x^m | \Theta) | x^g, \Theta^t]$ 
8      $\Theta^{t+1}$ <-  $\arg \max_{\Theta} Q(\Theta, \Theta^t)$ 
9     t = t+1
10  until que se alcanzado el criterio de convergencia
11 EndProcess

```

5. Ejecutar casos de prueba para regresión

El algoritmo N.º 5 presenta la fase final del enfoque que ejecuta los casos de prueba que pertenecen a un determinado clúster en el que los casos de prueba de la funcionalidad f_i haya fallado (TestDBfailed), como resultado se muestra la lista de fallas detectadas.

Algoritmo N.º 5: executeClusterRegressionTestDB

Input: MatClusterRegressionTestDB

Output: Listfailures

```

1 Process executeClusterRegressionTestDB
2   ListFailure <- ∅
3   for each  $Tc_i$  in TestDBfailed
4     execute(MatClusterTestDB  $\cap$  TestDBfailed)
5     ListFailures<- ListFailures  $\cup$   $tc_i$ 
6   end for
7 EndProcess
```

A continuación, se muestran otros algoritmos adicionales (algoritmo 6 BuildMatrixTestDB y algoritmo 7 AppendMatrix) que se diseñaron a fin de complementar el algoritmo principal, cuya implementación se denomina GENERADOR, que tiene como función principal separar los casos de prueba con acceso a datos y contrastarlos con el esquema de la base de datos e ir construyendo incrementalmente la matriz de ceros y unos para el proceso de clusterización.

Para el proceso de clusterización, esta matriz de ceros y unos está considerada como elemento de agrupación no supervisada utilizando la herramienta WEKA.

Algoritmo N.º 6 :BuildMatrixTestDB

```

Input:  RCD <- Route Clases Directory // call sytem
        RTD <- Set Test Clases in Directory Route // call system
        HDF <- Date_Hour File //Date and Hour call system
Output: Matrix Testcases


---


1: BuildMatrixTestDBprocess
2:   foreach classfile in RCD
3:     testClasefile <- generatorTestClases(classfile) //call JUNIT library
4:     if (INSERT|DELETE|UPDATE|SELECT) in classfile then
5:       testClasefile <- testClasefile U {header instructions line's test method}
6:       testClasefile <- testClasefile U {body instructions lines test depending type result}
7:       testClasefile <- testClasefile U {instructions lines test result}
8:       RTD <- RTD U testClasefile
9:       AppendMatrix(HDF, testClasefile)
10:    end if
11:  end foreach
12: end BuildMatrixTestDB process

```

A continuación, el algoritmo de AppendMatrix que genera la matriz de ceros y unos incrementalmente.

Algoritmo N.º 7: AppendMatrix

Input: testClase, HDF

 Output: file HDF

```

1: AppendMatrix process
2:   L <- Obtain columns and tables of DataBase // calls DataBase Manager System
3:   foreach testClassfile in HDF
4:     ISQL <- Obtain Instructions SQL of testClassfile
5:     if "DELETE" in ISQL then
6:       put 1 in columns of table selected // Filtered L
7:     end if
8:     if "INSERT" in ISQL
9:       put 1 in columns of table selected // Filtered L
10:    end if
11:    if "SELECT" in ISQL
12:      tmpISQL <- parser ISQL // separated tokens ISQL delete part WHERE
13:      if "INNER" join in tmpISQL then
14:        tmpISQL <- parser tmpISQL // separated each inner join
15:      end if
16:      foreach table of tmpISQL
17:        if "*" in ISQL then
18:          put 1 in columns of table selected // Filtered L
19:        else
20:          put 1 in columns of table selected where table.column == columnDB // Filtered L
21:        end if
22:      end foreach
23:    end if
24:    if "UPDATE" in ISQL then
25:      tmpISQL <- parser ISQL // separated tokens ISQL
26:      tmpcolumns <- list columns in tmpISQL
27:      foreach tmpcolumns
28:        put "1" in columns of table selected where table.column == columnDB // Filtered L
29:      end foreach
30:    end if
31:  end foreach
32: end AppendMatrix process

```

4.4. Supuestos

En esta sección se presenta las características sobre las cuales se fundamenta la consecución del aporte.

- Los tiempos de ejecución de los casos de pruebas son iguales.
- La severidad de los casos de prueba es igual.

- La capacidad de detección de falla (recall) de los casos de prueba es uno a uno.
- El entorno de desarrollo es iterativo incremental.
- Ambiente de desarrollo es Java con tecnología de pruebas unitarias.
- El tipo de base de datos es relacional.

4.5. Alcance

Método propuesto puede ser aplicado en proyectos que consideren ciclos de vida del software bajo ambientes de desarrollo iterativo incremental y que cuenten con tecnología de desarrollo de casos de prueba por cada una de las versiones que hayan sido implementadas.

4.6. Limitaciones

Su entorno de desarrollo es bajo la tecnología de Java con base de datos relacionales y pruebas unitarias; el diseño arquitectónico es modelo-vista-controlador (MVC)

CAPÍTULO V: ESTUDIO EMPÍRICO

El proceso software involucra desarrollo, operación y mantenimiento de productos software, así como de los artefactos relacionados. La investigación en ingeniería de software se enfoca a la búsqueda de cómo su desarrollo, operación y mantenimiento son conducidos por los ingenieros de software juntamente con los interesados bajo diferentes condiciones. El desarrollo de software es realizado, por lo tanto, por individuos, grupos u organizaciones, además, son afectadas por cuestiones sociales y políticas. (Kitchenham *et al.*, 1995)

En esta investigación se aplica el proceso de experimentación aplicado a la ingeniería de software denominado Ingeniería de software experimental (Natalia Juristo y Moreno, 2013; Kitchenham *et al.*, 2009); con 32 de 37 pruebas de regresión provenientes de cinco proyectos de software con acceso a datos desarrollados en entorno iterativo incremental.

De manera general el proceso de experimentación en ingeniería de software puede dividirse en cinco etapas principales:

1. Definición y diseño: En esta etapa se realiza la definición de objetivos, planteamiento de la(s) pregunta(s) de investigación o las hipótesis y se realiza la planeación de este.

2. Preparación para recolectar datos: Se define los procedimientos y protocolos para llevar a cabo el experimento para recolectar los datos.
3. Recolección de evidencia: Aquí la ejecución del caso de estudio y recolección de datos tiene lugar.
4. Análisis sobre los datos recolectados.
5. Reporte de resultados

5.1. Definición y diseño

Es la actividad inicial y que corresponde con las actividades de identificar aspectos como determinar el objeto de estudio, propósito, enfoque de calidad, perspectiva y contexto donde se realiza el experimento.

El objetivo general de este estudio es determinar de qué manera un método de clusterización probabilística permite mejorar las pruebas de regresión de software en entornos de desarrollo iterativo-incremental; el estudio se realizará desde la perspectiva del investigador, en el contexto del desarrollo de software con acceso a datos y bajo entornos de desarrollo iterativo-incremental.

Tres hipótesis específicas se plantean para contrastar las métricas/indicadores definidos y generados en la ejecución y recolección de datos de las pruebas de regresión ejecutadas.

- H1: El método de clusterización probabilística reduce la cantidad de casos de prueba en las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental.
- H2: El método de clusterización probabilística mejora la precisión de las pruebas de regresión en software con acceso a datos entornos de desarrollo iterativo incremental.
- H3: El método de clusterización probabilística mejora el rendimiento combinado de las pruebas de regresión en software con acceso a datos en entornos de desarrollo iterativo incremental.

Con respecto a las unidades de análisis, se consideran tres unidades de análisis de manera individual y luego de manera embebida, un caso de estudio embebido es aquel donde se analizan múltiples unidades de análisis: tres proyectos de software con acceso a datos de bajo y mediana escala, denominados: Estafeta, Silabo, Académico, GICO y Yellou.

5.2. Procedimiento y protocolos

5.2.1. Configuración

Continuando el proceso, se definen las metodologías, métricas y protocolos para la recolección de datos. Con respecto a las metodologías y técnicas (ver Tabla 12), se consideró la metodología de prueba de regresión por selección y dentro de ella a dos técnicas: prueba de regresión para software con acceso a datos en entorno iterativo incremental (enfoque propuesto) y la técnica de prueba de regresión total (RetestALL), que en esta investigación sirve como una técnica de control. RetestALL es una técnica de

práctica común, cuya característica principal es ejecutar todos los casos de prueba cada vez que tiene lugar una prueba de regresión.

Tabla 12. Metodologías y técnicas

Metodologías de prueba de regresión por selección	Método de clusterización probabilística de casos de prueba de prueba de regresión de software con acceso a datos
	Método de prueba de regresión convencional – regresión total (RetestAll)

5.2.2. Variables e indicadores

En esta investigación se considera como variable Independiente al método de clusterización probabilística y como variable dependiente a las pruebas de regresión de software con acceso a datos en entornos de desarrollo iterativo incremental, en este contexto los indicadores son las siguientes: cantidad de casos de prueba, precisión, capacidad de detección de fallas y rendimiento combinado, mismas que se han convertido en estándar para medir la eficiencia global en técnicas de pruebas regresión (G. Rothermel et al., 1998).

A continuación, se presenta las ecuaciones que sirven para recolectar y calcular los datos, manteniendo sus denominaciones por la literatura en inglés: capacidad de reducción como CR, cantidad de casos de prueba como TR (Test Reduccion por sus siglas en inglés), a precisión como Precision, capacidad de detección de fallas como Recuperación y la armónica de efectividad del método basado en la relación entre Precision y Recuperación como Rendimiento combinado:

- **Capacidad de reducción (TR).** – Sea T el número total de casos de prueba con acceso a la base de datos y T' es el número de casos de prueba seleccionados para una regresión, entonces TR viene dado por:

Ecuación 1. Indicador: Capacidad de reducción

$$TR = \frac{T-T'}{T} \times 100 \quad (1)$$

- **Cantidad de casos de prueba reducidos (CP).** – Es el número de casos de prueba con acceso a la base de datos relevantes para una prueba de regresión T'.

- **Precisión (P).** – Sea $T'f$ el conjunto de todos los casos de prueba seleccionados con acceso a la BD que revelan fallas, entonces P viene dado por:

Ecuación 2. Indicador: Precisión

$$\text{Precision} = \frac{T'f}{T'} \quad (2)$$

- **Recuperación (R).** – Sea Tf el conjunto de casos de prueba que revelan fallas, entonces la métrica de R viene dada por:

Ecuación 3 Indicador: Recuperación

$$\text{Recuperación} = \frac{|T'f|}{|Tf|} \quad (3)$$

La ecuación (3) es usada para determinar la capacidad de detección de fallos de los casos de prueba seleccionados, que, en esta investigación por diseño del método, en el paso número 5, siempre será del 100%, pues se seleccionan todos los clústeres que contienen todos los casos de prueba con fallos.

- **Rendimiento combinado** – Es una combinación de dos métricas ampliamente usadas en ciencias de la información, precisión (P) y recall (R), evalúa su beneficio de manera integral y está dada por la siguiente ecuación:

Ecuación 4. Indicador: Rendimiento combinado

$$\text{Rendimiento combinado} = \frac{2 \times \text{Precisión} \times \text{Recall}}{\text{Precisión} + \text{Recall}} \quad (4)$$

5.2.3. Unidades de análisis

Para la evaluación del método se desarrollaron 32 pruebas de regresión de 37 versiones de cinco productos software: (pequeña y mediana escala) con acceso a datos, bajo entorno Java: Estafeta, Silabo y Académico, GICO y Yellou, los cuales son utilizadas en un contexto de administración académico y empresarial. El primero denominado “Estafeta” que es un software para la gestión de la jornada laboral del profesor universitario, el segundo “Silabo” utilizado para la gestión de contenidos de las asignaturas que imparte un profesor en la Escuela Superior Politécnica de Chimborazo (ESPOCH), el tercero que permite administrar la componente escolástica de una entidad académica, el cuarto Gestor de inventarios de una empresa comercializadora y por último Yellou que es un software de dominio comercial similar al UBER, que por razones de propiedad intelectual.

Hay que recalcar que los productos de software con acceso a datos son conocidos como sistemas de información, ERP y CRM, por lo que bien puede usarse cualquier producto software que cuenten con acceso a datos a través de sistemas de bases de datos relacionales (Guamán, Rocío, Zabala, Cecibel, 2016 ;Broncano y Alexander, 2016; Ati y Fabricio, 2017; Díaz y Ortiz, 2018).

A continuación, en detalle la caracterización técnica, arquitectónica y funcional de los productos de software.

a. Estafeta

El producto software Estafeta es una aplicación cuya finalidad es la gestión de las agendas de actividades de los docentes investigadores de la ESPOCH, ellas deben contemplar cuatro componentes: docencia, investigación, vinculación y gestión, las mismas que deben cubrirse en una agenda de 40 horas semanales con algunas reglamentaciones específicas para cada una de ellas, por ejemplo, que el componente de docencia no puede superar las 32 horas a la semana.

A continuación, las características funcionales del producto Estafeta:

- Gestión de la Docencia: dicción de clases, preparación de clases, preparación de exámenes y calificaciones.
- Gestión de Investigación: participación en proyectos de investigación, publicación de artículos, publicación de libros, participación en eventos de transferencia de ciencia y tecnología, entre otros.
- Gestión Vinculación: actividades de vinculación con la colectividad a través de proyectos de vinculación.
- Gestión de apoyo a la Gestión: ejercer funciones de gestión, participación en actividades de gestión de carreras académicas, actividades de gestión de apoyo a los diferentes estamentos de gestión a nivel institucional.

La agenda de actividades se registra en el software “Estafeta” al inicio del periodo académico y sirve como un indicador de gestión de los docentes

de las unidades académicas de la ESPOCH, así como de la escuela(s) donde el docente realiza su labor.

A continuación, la Tabla 13 presenta las características técnicas del producto Estafeta:

Tabla 13. Características del producto software Estafeta

Producto software	Características	
Estafeta	N.º LOC	57,797
	BD.	Relacional
	Entidades	39
	Relaciones	92
	Promedio de Atributos x entidad/Relación	10
	N.º Test case con acceso a BD	200
	Plataforma de desarrollo	NetBeans
	Librerías	JUnit y DBUnit
	N.º Versiones	5

Desde el punto de vista arquitectónico, Estafeta es diseñado siguiendo la arquitectura de tres capas: capa de interfaz de usuarios, capa de lógica del negocio y la capa de acceso a datos, denominado Modelo Vista Controlador (MVC), donde el acceso a los datos es a través es la capa de controlador.

b. Silabo

El producto Silabo es un software que permite la gestión de los contenidos curriculares de las asignaturas de las carreras de las escuelas de una facultad dentro de la ESPOCH.

La gestión de contenidos considera aspectos microcurriculares como el plan analítico, planificación semestral y planificación semanal, cada una de las actividades se encuentra correlacionado con el software Estafeta por el tema

de dicción de clases, así como también la rendición de exámenes que así mismo están racionados con la agenda académica institucional.

Dentro de sus funcionalidades se considera:

- Gestión plan de contenidos mesocurriculares de las asignaturas.
- Gestión de contenidos microcurriculares de las asignaturas.
- Gestión de registro de dicción de clases por componentes microcurriculares.

Al igual que el software anterior, sirve como indicador de gestión docentes en las carreras de la ESPOCH.

Dentro del aspecto técnico de Silabo, a continuación, la Tabla 14 presenta las características técnicas de este software.

Tabla 14. Características del producto software Silabo

Producto software	Características	
Silabo	N.º LOC	35,880
	BD.	Relacional
	Entidades	42
	Relaciones	88
	Promedio de Atributos x entidad/Relación	10
	N.º Test case con acceso a BD	300
	Plataforma de desarrollo	NetBeans
	Librerías	JUnit y DBUnit
	N.º Versiones	9

Desde el punto de vista arquitectónico Estafeta es diseñado siguiendo la arquitectura de tres capas, denominado Modelo Vista Controlador (MVC).

c. Académico

El producto software denominado Académico es un software cuya finalidad es la gestión macrocurricular de la ESPOCH, es decir, gestiona los aspectos relacionados a la gestión de facultades y carreras.

Dentro de esas actividades se pueden considerar:

- Gestión de carreras
- Gestión mallas curriculares
- Gestión de periodos académicos
- Gestión de matrículas

Al igual que los anteriores productos de software, sirven como indicador de gestión académica.

Dentro del aspecto técnico de Académico, a continuación, la Tabla 15 presenta las características técnicas de este software.

Tabla 15. Características del producto software Académico

Producto software	Características	
Académico	N.º LOC	81,150
	BD.	Relacional
	Entidades	155
	Relaciones	375
	Promedio de Atributos x entidad/Relación	12
	N.º Test case con acceso a BD	600
	Plataforma de desarrollo	NetBeans
	Librerías	JUnit y DBUnit
	N.º Versiones	12

Desde el punto de vista arquitectónico, Académico es diseñado siguiendo la arquitectura de tres capas, denominado Modelo Vista Controlador (MVC).

d. GICO

El producto software de gestión de procesos de una empresa comercializadora de productos de primera necesidad, entre sus procesos se cuenta con la gestión de proveedores, clientes, pedidos, compras, ventas, reportes e inventarios

A continuación, las características funcionales del producto:

- Gestión de pedidos, Gestión de compras, Gestión inventario, Gestión ventas, Gestión proveedores, Gestión de clientes, Reportes.

A continuación, se presenta la Tabla 16 con las características técnicas del producto.

Tabla 16. Características del producto software GICO

Producto software	Características	
GICO	N.º LOC	35.000
	BD.	Relacional
	Entidades	20
	Relaciones	15
	Promedio de Atributos x entidad/Relación	8
	N.º Test case con acceso a BD	175
	Plataforma de desarrollo	NetBeans
	Librerías	Junit, AJAX y DBUnit
	N.º Versiones	5

Desde el punto de vista arquitectónico, GICO es diseñado siguiendo la arquitectura de tres capas: capa de interfaz de usuarios, capa de lógica del negocio y la capa de acceso a datos, denominado Modelo Vista Controlador (MVC), donde el acceso a los datos es a través es la capa de controlador.

e. Yellow

El producto Yellow es un software que permite la gestión de los procesos de negocios multilaterales de una empresa de una red de transporte a nivel de la ciudad de Riobamba, con visión nacional, que conecta a quienes conducen y realizan entregas a usuarios, clientes y restaurantes, negocio similar a UBER.

Dentro de sus funcionalidades se considera:

- Gestión bancos, Gestión de cartera, Gestión de facturación, Gestión empresas, Gestión reportes transacciones, gestión de seguridad, Gestión de conductores.

A continuación, se presenta la Tabla 17 con las características técnicas del producto

Tabla 17. Características del producto software Yellow

Producto software	Características	
YELLOW	N.º LOC	350.000
	BD.	Relacional
	Entidades	38
	Relaciones	45
	Promedio de Atributos x entidad/Relación	8
	N.º Test case con acceso a BD	675
	Plataforma de desarrollo	NetBeans
	Librerías	JUnit y DBUnit
	N.º Versiones	6

Desde el punto de vista arquitectónico YELLOW esta diseñado siguiendo la arquitectura de tres capas, denominado Modelo Vista Controlador (MVC).

5.3. Ejecución y recopilación de evidencia

Como se mencionó anteriormente, el método de clusterización probabilístico es utilizado para agrupar los casos de prueba basados en su similitud de relación de acción sobre los campos de las tablas de una base de datos relacional.

Para el primer producto (Estafeta) se desarrollaron cinco versiones, cada una con incremento de casos de prueba, para el segundo producto (Silabo) se desarrollaron nueve versiones, y para el tercero (Académico) doce versiones, para el cuarto (GICO) cinco versiones y por último Yellow 6 versiones; los cinco proyectos siguen la metodología de desarrollo iterativo incremental, es decir que en cada versión se añaden nuevas funcionalidades y que, por lo tanto, también se diseñan y desarrollan nuevos casos de prueba; lo anterior sigue la metodología y herramientas para prueba unitarias bajo Java tanto para el código como para la Base de Datos (JUnit, DBUnit). Por otro lado, la información de los casos de prueba y la documentación de los planes de prueba se encuentran en los servicios web de la nube de Amazon AWS ([http:// raulrosero.ml](http://raulrosero.ml)).

5.3.1. Planes de pruebas

Con el objetivo de documentar los casos de prueba unitarios que validan y verifican que los productos software satisfacen las funcionalidades especificadas, se desarrolló el plan de pruebas unitarias de cada producto software, cada plan de pruebas contiene la definición de los casos de prueba, sus requisitos, y la estrategia a seguir en la ejecución de las pruebas.

Las funcionalidades de Estafeta, Silabo, Académico y GICO se encuentran detalladas en las respectivas tesis de grado mientras que Yellou se mantiene en reserva dado su característica de proyecto comercial, por lo que el plan de pruebas se ajusta a las versiones consideradas para pruebas de regresión de este trabajo.

En consideración que la documentación de los planes de prueba de los productos software son extensos (1950 casos de prueba), se utilizó TestLink (<http://testlink.org>), que es un software de Gestión de Planes de Prueba. El link para utilización es <http://raulrosero.ml> con las siguientes credenciales: user=admin y password=admin. En el Anexo B se muestra de manera representativa el plan de pruebas de la especificación 1 del módulo 1 con sus casos de prueba del sistema Académico.

5.3.2. Instrumentación adicional

Para el procedimiento para la recolección de datos, se diseñaron y desarrollaron herramientas de software adicionales, que sirven para la construcción de la matriz de ceros y unos, la misma que sirve como insumo para el proceso de agrupación. A continuación, los algoritmos que se consideran en esta herramienta:

BuildMatrixTestDB.– El objetivo de este algoritmo (algoritmo 6) es analizar los casos de prueba del proyecto, este análisis lo hace en el contexto de la herramienta de desarrollo para JAVA, denominada NetBeans, a continuación selecciona cada caso de prueba que contenga código de acceso a datos, es decir, que contenga operaciones de inserción, eliminación, actualización y consulta a los campos de las tablas de la base de datos.

Una vez que tiene el código de acceso a la data, el algoritmo realiza una operación de confrontación con el esquema de la base de datos a fin de obtener los campos y tablas sobre los cuales la operación tiene lugar (Algoritmo 7 AppendMatrix). Finalmente, para terminar de este proceso, se construye una matriz de ceros y unos, donde los unos significan una acción sobre ese campo, caso contrario se observa con ceros.

Para el proceso de agrupación, esta matriz de ceros y unos está considerada como elemento de agrupación no supervisada utilizando la herramienta WEKA.

Algoritmo 6 :BuildMatrixTestDB

Input: RCD <- Route Clases Directory // call sytem
 RTD <- Set Test Clases in Directory Route // call system
 HDF <- Date_Hour File //Date and Hour call system

Output: Matrix Testcases

```

1: BuildMatrixTestDBprocess
2:   foreach classfile in RCD
3:     testClasefile <- generatorTestClases(classfile) //call JUNIT library
4:     if (INSERT|DELETE|UPDATE|SELECT) in classfile then
5:       testClasefile <- testClasefile U {header instructions line's test method}
6:       testClasefile <- testClasefile U {body instructions lines test depending type
result}
7:       testClasefile <- testClasefile U {instructions lines test result}
8:       RTD <- RTD U testClasefile
9:       AppendMatrix(HDF, testClasefile)
10:    end if
11:  end foreach
12: end BuildMatrixTestDB process

```

Algoritmo 7: AppendMatrix

Input: testClase, HDF

Output: file HDF

```

1: AppendMatrix process
2:   L <- Obtain columns and tables of DataBase // calls DataBase Manager System
3:   foreach testClassfile in HDF
4:     ISQL <- Obtain Instructions SQL of testClassfile
5:     if "DELETE" in ISQL then
6:       put 1 in columns of table selected // Filtered L
7:     end if
8:     if "INSERT" in ISQL
9:       put 1 in columns of table selected // Filtered L
10:    end if
11:    if "SELECT" in ISQL
12:      tmpISQL <- parser ISQL // separated tokens ISQL delete part WHERE
13:      if "INNER" join in tmpISQL then
14:        tmpISQL <- parser tmpISQL // separated each inner join
15:      end if
16:      foreach table of tmpISQL
17:        if "*" in ISQL then
18:          put 1 in columns of table selected // Filtered L
19:        else
20:          put 1 in columns of table selected where table.column == columnDB // Filtered L
21:        end if
22:      end foreach
23:    end if
24:    if "UPDATE" in ISQL then
25:      tmpISQL <- parser ISQL //      separated tokens ISQL
26:      tmpcolumns <- list columns in tmpISQL
27:      foreach tmpcolumns
28:        put "1" in columns of table selected where table.column == columnDB // Filtered L
29:      end foreach
30:    end if
31:  end foreach
32: end AppendMatrix process

```

5.3.3. Recopilación de evidencia

A continuación, los datos de cada versión en cada uno de los productos software considerados en esta investigación. La Tabla 18 presenta los casos de prueba nuevos por cada versión en cada uno de los productos, así como los casos de prueba para una prueba de regresión.

Tabla 18. Número de casos de prueba por versiones en cada producto software

Producto	Versión	Casos de prueba (Tn)	Casos de prueba para regresión (T)
Estafeta	1	29	-
	2	43	72
	3	62	134
	4	29	163
	5	36	199
Silabo	1	27	-
	2	33	60
	3	29	89
	4	30	119
	5	39	158
	6	49	207
	7	32	239
	8	22	261
	9	39	300
Académico	1	45	-
	2	47	92
	3	72	164
	4	41	205
	5	40	245
	6	57	302
	7	54	356
	8	50	406
	9	47	453
	10	27	480
	11	48	528
	12	72	600
GICO	1	40	-
	2	41	81
	3	39	120
	4	27	147
	5	29	176
Yellou	1	140	-
	2	150	290
	3	105	395
	4	165	560
	5	50	610
	6	65	675

Para cada uno de los productos se ejecutaron los respectivos casos de prueba de cada una de las versiones, luego se realizó una prueba de regresión del producto en cada versión.

La Tabla 19 reporta los fallos (Tf) que fueron identificados para la prueba de regresión por producto software, considerando que en la versión 1 no se realizó una prueba de regresión.

Tabla 19. Casos de prueba que fallan en cada versión de producto software

Productos software	Versión	Casos de prueba (T)	fallos (Tf)
Estafeta	2	72	8
	3	134	11
	4	163	9
	5	199	10
Silabo	2	60	8
	3	89	10
	4	119	15
	5	158	5
	6	207	7
	7	239	10
	8	261	5
	9	300	4
Académico	2	92	6
	3	164	5
	4	205	6
	5	245	4
	6	302	6
	7	356	4
	8	406	6
	9	453	4
	10	480	3
	11	528	5
	12	600	4
GICO	2	81	13
	3	120	5
	4	147	10
	5	176	15
Yellou	2	290	30
	3	395	13
	4	560	27
	5	610	40
	6	675	7

En esta investigación se usaron fallos de regresión reales para evaluar el método propuesto, además, se considera que la severidad, tiempo de ejecución y los patrones de fallos de los programadores son iguales.

Para la fase de agrupación se utilizó el algoritmo probabilístico Expectation Maximization (EM), que es una versión extendida de K-means, en donde la estimación de los parámetros del modelo es optimizada. EM usa el método de asignación suave para asignación de elementos a un grupo (el elemento se puede agrupar en varios grupos), mientras que otros algoritmos usan asignación dura (un elemento pertenece a un solo y solo un grupo, por ejemplo K-means) (Kung, 2014). El algoritmo EM determina el parámetro k = número de grupos mediante una validación cruzada de la información suministrada (algoritmo N.º 4), el máximo número de iteraciones en este trabajo se realizó con 1×10^{-6} como el mínimo valor desviación estándar para converger y el número máximo de iteraciones en 10 con varias semillas. (Witten y Frank, 2005)

En este trabajo, a fin de reducir el impacto aleatorio de la determinación aleatoria del centroide que constituye una debilidad en los algoritmos de agrupación, se ha considerado el trabajo de optimización de Celeux & Govaert (1992) y Hamerly & Elkan (2002), en los que trabajo temas de centroide básicamente.

Se realizó un trabajo estadístico en cada versión con 10 semillas diferentes (1, 10, 50, 100, 150, 200, 250, 500, 750, 1000), obteniendo mediciones para cada una de las variables consideradas en este estudio, para posteriormente realizar un análisis de los estadígrafos (media, varianza, desviación estándar) en cada una de las versiones de los productos software. A manera de representación de los datos, con mejor desempeño se presentan las Tablas 20, 21, 22, 23 y 24 obtenidas en las versiones 4, 7, 5, 3 y 5 de los productos software Estafeta, Silabo, Académico, GICO y Yelloou respectivamente, remarcando que el parámetro k lo determinó el algoritmo EM de Weka.

Tabla 20. Clusterización versión 4 de Estafeta con varias semillas

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	3	163	29	148	9	9	9.2%	100%	6%	0.11
10	3	163	29	147	9	9	9.8%	100%	6%	0.12
50	3	163	29	122	9	9	25.2%	100%	7%	0.14
100	4	163	29	33	9	9	79.8%	100%	27%	0.43
150	4	163	29	154	9	9	5.5%	100%	6%	0.11
200	7	163	29	121	9	9	25.8%	100%	7%	0.14
250	6	163	29	134	9	9	17.8%	100%	7%	0.13
500	5	163	29	154	9	9	5.5%	100%	6%	0.11
750	6	163	29	147	9	9	9.8%	100%	6%	0.12
1000	7	163	29	148	9	9	9.2%	100%	6%	0.11

Tabla 21. Clusterización versión 7 de Silabo con varias semillas

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	9	239	32	31	9	9	87.0%	100%	29%	0.45
10	7	239	32	180	9	9	24.7%	100%	5%	0.10
50	3	239	32	195	9	9	18.4%	100%	5%	0.09
100	9	239	32	31	9	9	87.0%	100%	29%	0.45
150	4	239	32	40	9	9	83.3%	100%	23%	0.37
200	7	239	32	170	9	9	28.9%	100%	5%	0.10
250	5	239	32	41	9	9	82.8%	100%	22%	0.36
500	5	239	32	40	9	9	83.3%	100%	23%	0.37
750	7	239	32	171	9	9	28.5%	100%	5%	0.10
1000	6	239	32	203	9	9	15.1%	100%	4%	0.08

Tabla 22. Clusterización versión 5 de Académico con varias semillas

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	4	245	40	131	4	4	46.5%	100%	3%	0.06
10	3	245	40	12	4	4	95.1%	100%	33%	0.50
50	5	245	40	45	4	4	81.6%	100%	9%	0.16
100	4	245	40	11	4	4	95.5%	100%	36%	0.53
150	3	245	40	121	4	4	50.6%	100%	3%	0.06
200	5	245	40	45	4	4	81.6%	100%	9%	0.16
250	4	245	40	121	4	4	50.6%	100%	3%	0.06
500	2	245	40	11	4	4	95.5%	100%	36%	0.53
750	3	245	40	25	4	4	89.8%	100%	16%	0.28
1000	4	245	40	89	4	4	63.7%	100%	4%	0.09

Tabla 23. Clusterización versión 5 de GICO con varias semillas

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	3	176	29	54	7	7	69.3%	100%	13%	0.23
10	4	176	29	123	7	7	30.1%	100%	6%	0.11
50	5	176	29	56	7	7	68.2%	100%	13%	0.22
100	4	176	29	145	7	7	17.6%	100%	5%	0.09
150	5	176	29	135	7	7	23.3%	100%	5%	0.10
200	5	176	29	24	7	7	86.4%	100%	29%	0.45
250	3	176	29	167	7	7	5.1%	100%	4%	0.08
500	4	176	29	100	7	7	43.2%	100%	7%	0.13
750	4	176	29	159	7	7	9.7%	100%	4%	0.08
1000	3	176	29	98	7	7	44.3%	100%	7%	0.13

Tabla 24. Clusterización versión 3 de Yellow con varias semillas

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	4	395	105	200	13	13	49.4%	100%	7%	0.12
10	6	395	105	133	13	13	66.3%	100%	10%	0.18
50	3	395	105	145	13	13	63.3%	100%	9%	0.16
100	5	395	105	144	13	13	63.5%	100%	9%	0.17
150	5	395	105	157	13	13	60.3%	100%	8%	0.15
200	6	395	105	126	13	13	68.1%	100%	10%	0.19
250	5	395	105	136	13	13	65.6%	100%	10%	0.17
500	3	395	105	225	13	13	43.0%	100%	6%	0.11
750	4	395	105	120	13	13	69.6%	100%	11%	0.20
1000	4	395	105	290	13	13	26.6%	100%	4%	0.09

CAPITULO VI: ANÁLISIS DE DATOS

6.1. Análisis del impacto de fallos en las pruebas de regresión

A continuación, se presenta los valores resumidos del impacto de fallos obtenidos en cada una de las ejecuciones de clusterización EM de cada prueba de regresión de cada proyecto.

a) Producto Software: Estafeta

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante los métodos considerados en las diferentes versiones de Estafeta, a continuación, la Tabla 25 con el resumen de los datos obtenidos, en los grupos de control y de experimentación:

Tabla 25. Comparativo global de las técnicas en el producto software Estafeta

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	14%	0%
Recuperación	100%	100%
Precision	10%	7%
Rendimiento combinado	0.17	0,11

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización, con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima

al valor recomendado de 1 del rendimiento combinado, considerando la capacidad de reducción de casos de prueba del 14%

b) Producto Software: Silabo

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante las técnicas consideradas, en las diferentes versiones de Silabo, a continuación, la Tabla 26 con el resumen de los datos obtenidos, en los grupos de control y de experimentación:

Tabla 26. Comparativo global de las técnicas en el producto software Silabo

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	40%	0%
Recuperación	100%	100%
Precision	12%	6%
Rendimiento combinado	0.20	0,09

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización, con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima al valor recomendado de 1 de la armónica F-measure, considerando la capacidad de reducción del 40%.

c) Producto Software: Académico

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante las técnicas consideradas, en las diferentes versiones de Silabo, a continuación, la Tabla 27 con el resumen de los datos obtenidos, en los grupos de control y de experimentación:

Tabla 27. Comparativo global de las técnicas en el producto software Académico

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	36%	0%
Recuperación	100%	100%
Precision	6%	1%
Rendimiento combinado	0.10	0,01

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización, con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima al valor recomendado de 1 de la armónica F-measure, considerando la capacidad de reducción del 36%.

d) Producto Software GICO

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante las técnicas consideradas, en las diferentes versiones de Silabo, a continuación, la Tabla 28 con el resumen de los datos obtenidos, en los grupos de control y de experimentación:

Tabla 28 Comparativo global de los métodos en el producto GICO

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	43%	0%
Recuperación	100%	100%
Precision	16%	8%
Rendimiento combinado	26%	14%

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización,

con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima al valor recomendado de 1 del rendimiento global, considerando la capacidad de reducción de casos de prueba del 43%

e) Producto Software YELLOU

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante las técnicas consideradas, en las diferentes versiones de Silabo, a continuación, la Tabla 29 con el resumen de los datos obtenidos, en los grupos de control y de experimentación:

Tabla 29 Comparativo global de las técnicas en el producto software YELLOU

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	51%	0%
Recuperación (Recall)	100%	100%
Precision	12%	5%
Rendimiento combinado	21%	10%

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización, con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima al valor recomendado de 1 del rendimiento combinando, considerando la capacidad de reducción de casos de prueba del 51%.

La siguiente Tabla 30 muestra un resumen general de los promedios de los indicadores de los cinco proyectos de software, haciendo notar que de manera global el indicador de recuperación (recall) en los cinco casos es del 100%, con una capacidad de reducción de la cantidad de casos de prueba

promedio del 37% frente a una precisión promedio del 11%. Lo que implica un rendimiento combinado de un 19% del método.

Tabla 30. Estadígrafos Resumen de cada producto software

Indicadores	ESTAFETA	SILABO	ACADEMICO	GICO	YELLOW
Cantidad de casos de prueba	14%	40%	36	43%	51%
Recuperación (Recall)	100%	100%	100	100%	100%
Precision	10%	12%	6	16%	12%
Rendimiento combinado	17%	20%	10	26%	21%
#CP	199	309	600	176	675
#LDC	57K	35K	89K	35K	350K

6.2. Análisis de las pruebas de regresión por producto de software

Una vez obtenidos los valores de los indicadores respectivas, la siguiente actividad es realizar un análisis formal sobre ellas. En esta actividad se utilizan técnicas estadísticas de análisis de media y varianza para interpretar los valores. En esta investigación se aplicará una estrategia de análisis por indicador, considerando las pruebas de regresión de cada producto software y posteriormente determinar el impacto global del método propuesto.

A continuación, las Tablas 31, 32, 33, 34, 35 se muestran los estadígrafos descriptivos (media, desviación estándar, varianza y rango) de cada indicador de una prueba de regresión en cada una de las versiones de los productos software: Estafeta, Silabo, Académico, GICO y Yellow.

Tabla 31. Estadígrafos de cada versión del software Estafeta

Indicador/Estadígrafo	Media	Desv.Est.	Varianza	Rango
Versión 2				
Cantidad de casos de prueba	0.15	0.12	0.01	0.32
Recuperación (Recall)	1	0	0	0
Precision	0,13	0,20	0,00	0,05
Rendimiento combinado	0.23	0.03	0.00	0.08
Versión 3				
Cantidad de casos de prueba	0.06	0.05	0.00	0.12
Recuperación (Recall)	1	0	0	0
Precision	0.09	0.00	0.00	0,01
Rendimiento combinado	0.16	0.01	0.00	0.02
Versión 4				
Cantidad de casos de prueba	0.20	0.22	0.05	0.74
Recuperación (Recall)	1	0	0	0
Precision	0.09	0.07	0.00	0.21
Rendimiento combinado	0.15	0.10	0.01	0.32
Versión 5				
Cantidad de casos de prueba	0.16	0.27	0.07	0.81
Recuperación (Recall)	1	0	0	0
Precision	0.08	0,07	0.01	0.24
Rendimiento combinado	0.14	0.11	0.01	0.35
Promedio del Producto				
Cantidad de casos de prueba	0.14	0.19	0.03	0.82
Recuperación (Recall)	1	0	0	0
Precision	0.10	0.05	0.00	0.23
Rendimiento combinado	0.17	0.08	0.01	0.35

Se hace notar que en algunas mediciones (versiones) de los indicadores se obtuvieron valores altos que son contrarrestadas con valores bajo de otras corridas, siendo sensible los valores de k y de las semillas (seed) en ellas, además que la métrica de capacidad de detección de fallos es 100% por diseño del método. Bajo mismas consideraciones se presenta los datos de Silabo y Académico con sus indicadores.

Tabla 32. Estadígrafos de cada versión del software Silabo

Indicador/Estadígrafo	Media	Desv.Est.	Varianza	Rango
Versión 2				
Cantidad de casos de prueba	0.30	0.16	0.03	0.55
Recuperación (Recall)	1	0	0	0
Precision	0.20	0.07	0.00	0.23
Rendimiento combinado	0.33	0.09	0.01	0.29
Versión 3				
Cantidad de casos de prueba	0.26	0.24	0.06	0.71
Recuperación (Recall)	1	0	0	0
Precision	0.18	0.09	0.01	0.27
Rendimiento combinado	0.29	0.12	0.01	0.35
Versión 4				
Cantidad de casos de prueba	0.64	0.30	0.09	0.91
Recuperación (Recall)	1	0	0	0
Precision	0.18	0.08	0.01	0.22
Rendimiento combinado	0.29	0.13	0.02	0.32
Versión 5				
Cantidad de casos de prueba	0.49	0.30	0.09	0.70
Recuperación (Recall)	1	0	0	0
Precision	0.09	0.05	0.00	0.12
Rendimiento combinado	0.16	0.08	0.01	0.20
Versión 6				
Cantidad de casos de prueba	0.32	0.24	0.06	0.79
Recuperación (Recall)	1	0	0	0
Precision	0.06	0.05	0.00	0.16
Rendimiento combinado	0.12	0.08	0.01	0.27
Versión 7				
Cantidad de casos de prueba	0.54	0.33	0.11	0.72
Recuperación (Recall)	1	0	0	0
Precision	0.15	0.11	0.01	0.25
Rendimiento combinado	0.25	0.16	0.03	0.37
Versión 8				
Cantidad de casos de prueba	0.29	0.24	0.06	0.81
Recuperación (Recall)	1	0	0	0
Precision	0.05	0.09	0.01	0.29
Rendimiento combinado	0.09	0.14	0.02	0.43
Versión 9				
Cantidad de casos de prueba	0.37	0.24	0.06	0.91
Recuperación (Recall)	1	0	0	0
Precision	0.06	0.12	0.01	0.39
Rendimiento combinado	0.09	0.17	0.03	0.54
Promedio del producto				
Cantidad de casos de prueba	0.40	0.28	0.08	1.00
Recuperación (Recall)	1	0	0	0
Precision	0.12	0.10	0.01	0.39
Rendimiento combinado	0.20	0.15	0.02	0.54

Se hace notar que los valores de las métricas son mejores que estafeta, siempre considerando la sensibilidad de k y seed en el algoritmo de agrupación implementado en Weka.

Tabla 33. Estadígrafos de cada versión del software Académico

Métrica/Estadígrafo	Media	Desv.Est.	Varianza	Rango
Versión 2				
Cantidad de casos de prueba	0,2315	0,2238	0,0501	0,7283
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,1108	0,1019	0,0104	0,3268
Rendimiento combinado	0,1883	0,1352	0,0183	0,4351
Versión 3				
Cantidad de casos de prueba	0,1970	0,1363	0,0186	0,4390
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,03909	0,00747	0,00006	0,02386
Rendimiento combinado	0,07515	0,01375	0,00019	0,04392
Versión 4				
Cantidad de casos de prueba	0,1141	0,2227	0,0496	0,7171
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,03814	0,02307	0,00053	0,07418
Rendimiento combinado	0,0727	0,0406	0,0016	0,1306
Versión 5				
Cantidad de casos de prueba	0,7506	0,2022	0,0409	0,4898
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,1540	0,1434	0,0206	0,3331
Rendimiento combinado	0,2442	0,2030	0,0412	0,4741
Versión 6				
Cantidad de casos de prueba	0,1583	0,2324	0,0540	0,7483
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,02795	0,01822	0,00033	0,05908
Rendimiento combinado	0,0538	0,0331	0,0011	0,1074
Versión 7				
Cantidad de casos de prueba	0,2354	0,2317	0,0537	0,6938
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,01740	0,01056	0,00011	0,03400
Rendimiento combinado	0,03401	0,01996	0,00040	0,06424
Versión 8				
Cantidad de casos de prueba	0,518	0,406	0,165	0,879
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,0908	0,0852	0,0073	0,2243
Rendimiento combinado	0,1566	0,1396	0,0195	0,3562
Versión 9				
Cantidad de casos de prueba	0,410	0,355	0,126	0,887
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,02950	0,03096	0,00096	0,07803
Rendimiento combinado	0,0558	0,0564	0,0032	0,1423
Versión 10				
Cantidad de casos de prueba	0,6265	0,2821	0,0796	0,7000
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,0475	0,0471	0,0022	0,1028
Rendimiento combinado	0,0872	0,0846	0,0071	0,1836
Versión 11				
Cantidad de casos de prueba	0,385	0,404	0,163	0,920
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,0733	0,1003	0,0101	0,2174
Rendimiento combinado	0,1229	0,1626	0,0264	0,3509
Versión 12				

Métrica/Estadígrafo	Media	Desv.Est.	Varianza	Rango
Cantidad de casos de prueba	0,3167	0,2390	0,0571	0,6617
Recuperación (Recall)	1,0000	0,000000	0,000000	0,000000
Precision	0,01146	0,00580	0,00003	0,01811
Rendimiento combinado	0,02261	0,01125	0,00013	0,03507

Como se observa, el valor máximo de Capacidad de reducción de casos de prueba es 95,5 cuando el valor de la semilla (seed) es de 100 y el número de grupos es de $k=4$. En esta versión los valores promedios son óptimos alcanzando el valor promedio general de 75,1 de capacidad de reducción, 100% de capacidad de atrapar fallas y una precisión de 15,4%.

Tabla 34. Estadígrafos de cada versión del software GICO

Métrica/Estadígrafo	Media	Desv.Est.	Varianza	Rango
Versión 2				
Cantidad de casos de prueba	0.4988	0.1733	0.0300	0.6420
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.3625	0.1563	0.0244	0.5763
Rendimiento combinado	0.5172	0.1465	0.0215	0.5496
Versión 3				
Cantidad de casos de prueba	0.4258	0.1720	0.0296	0.4750
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.0811	0.0324	0.0011	0.0960
Rendimiento combinado	0.1485	0.0538	0.0029	0.1579
Versión 4				
Cantidad de casos de prueba	0.4122	0.1316	0.0173	0.4558
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.12113	0.02779	0.00077	0.09727
Rendimiento combinado	0.2151	0.0436	0.0019	0.1527
Versión 5				
Cantidad de casos de prueba	0.4241	0.2765	0.0764	0.8125
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.0964	0.0735	0.0054	0.2498
Rendimiento combinado	0.1691	0.1106	0.0122	0.3712

Se hace notar que en algunas mediciones (versiones) de los indicadores se obtuvieron valores altos que son contrarrestadas con valores bajo de otras corridas, siendo sensible los valores de k y de las semillas (seed) en ellas, además que la métrica de capacidad de detección de fallos es 100% por diseño del método propuesto.

Tabla 35. Estadígrafos de cada versión del software Yellou

Métrica/Estadígrafo	Media	Desv.Est.	Varianza	Rango
Versión 2				
Cantidad de casos de prueba	0.3890	0.1413	0.0200	0.3414
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.1771	0.0375	0.0014	0.0919
Rendimiento combinado	0.2993	0.0548	0.0030	0.1337
Versión 3				
Cantidad de casos de prueba	0.5757	0.1379	0.0190	0.4304
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.08352	0.02091	0.00044	0.06351
Rendimiento combinado	0.1535	0.0361	0.0013	0.1097
Versión 4				
Cantidad de casos de prueba	0.6077	0.1655	0.0274	0.4571
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.1417	0.0513	0.0026	0.1323
Rendimiento combinado	0.2450	0.0794	0.0063	0.2059
Versión 5				
Cantidad de casos de prueba	0.5125	0.2586	0.0669	0.6721
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.1786	0.1026	0.0105	0.3216
Rendimiento combinado	0.2921	0.1400	0.0196	0.4260
Versión 6				
Cantidad de casos de prueba	0.4422	0.2383	0.0568	0.8015
Recuperación (Recall)	1.0000	0.000000	0.000000	0.000000
Precision	0.02372	0.01479	0.00022	0.04302
Rendimiento combinado	0.04598	0.02774	0.00077	0.08083

Se hace notar que en algunas mediciones (versiones) de los indicadores se obtuvieron valores altos que son contrarrestadas con valores bajo de otras corridas, siendo sensible los valores de k y de las semillas (seed) en ellas, además que la métrica de capacidad de detección de fallos es 100% por diseño del método propuesto.

La siguiente Tabla 36 muestra un resumen general de los promedios de las métricas del indicador de eficiencia de los cinco proyectos de software, haciendo notar que de manera global el recall en los casos es del 100%, con una cantidad de casos de prueba en promedio del 36% frente a una precisión promedio del 9%. Lo que implica que su armónica de rendimiento combinado es de un 12%.

Tabla 36. Estadígrafos Resumen de cada producto software

Indicador	Estafeta	Silabo	Académico	GICO	Yellou
Cantidad de casos de prueba	14%	40%	36%	43%	51%
Recuperación (Recall)	100%	100%	100%	100%	100%
Precision	10%	12%	6%	16%	12%
Rendimiento combinado	17%	20%	10%	26%	21%
#CP	199	309	600	176	675
#LDC	57,797	35,880	89,000	35,000	350,000

6.3. Análisis de datos por indicador

A continuación, presenta los datos resultados de la experimentación.

Tabla 37 Indicadores por pruebas de regresión y grupos de analisis

Prueba de regresión	Cantidad de casos de prueba (número de casos de prueba)		% Precision (Porcentaje)		Rendimiento combinado (número)	
	Grupo de		Grupo de		Grupo de	
	experimental	control	experimental	control	experimental	control
PR1	61	72	0.13	0.11	0.23	0.20
PR2	126	134	0.09	0.08	0.16	0.15
PR3	131	163	0.08	0.06	0.15	0.14
PR4	168	199	0.08	0.05	0.14	0.15
PR5	42	60	0.21	0.13	0.34	0.24
PR6	66	89	0.18	0.11	0.30	0.20
PR7	45	119	0.18	0.05	0.29	0.10
PR8	80	158	0.09	0.03	0.17	0.06
PR9	141	207	0.07	0.03	0.12	0.07
PR10	110	239	0.13	0.04	0.22	0.07
PR11	186	261	0.06	0.02	0.10	0.04
PR12	189	300	0.06	0.01	0.10	0.03
PR13	71	92	0.11	0.07	0.19	0.12
PR14	132	164	0.04	0.03	0.08	0.06
PR15	182	205	0.04	0.03	0.07	0.06
PR16	61	245	0.16	0.02	0.24	0.03
PR17	254	302	0.03	0.02	0.05	0.04
PR18	272	356	0.02	0.01	0.03	0.02
PR19	196	406	0.09	0.01	0.16	0.03
PR20	267	453	0.03	0.01	0.06	0.01
PR21	179	480	0.05	0.01	0.09	0.01
PR22	325	528	0.07	0.01	0.12	0.02
PR23	410	600	0.01	0.01	0.02	0.01
PR24	41	81	1.06	0.16	0.52	0.28
PR25	69	120	1.10	0.04	0.15	0.08
PR26	86	147	0.26	0.07	0.22	0.13
PR27	106	176	0.32	0.04	0.16	0.08
PR28	97	290	0.18	0.10	0.30	0.19
PR29	168	395	0.08	0.03	0.15	0.06
PR30	220	560	0.14	0.05	0.24	0.09
PR31	297	610	0.18	0.07	0.29	0.12
PR32	377	675	0.02	0.01	0.05	0.02

El análisis de datos por indicador se realiza sobre el resumen del comportamiento de los indicadores en las pruebas de regresión de cada versión incremental de los productos de software en los grupos de análisis: grupo experimental y grupo de control, como se muestra en la Tabla 37.

f) Indicador: Cantidad de casos de prueba

La Tabla 37 muestra la cantidad de casos de prueba seleccionados en cada prueba de regresión para los grupos de análisis, por ejemplo, en la prueba de regresión dos se muestra la cantidad de casos de prueba considerados para la prueba de regresión en los grupos de análisis: grupo experimental 126 y grupo de control 134; lo que muestra una reducción de 8 (6%) casos de prueba con relación a 134 del grupo de control, en este caso, y que constituye el universo de casos de prueba, hasta esa prueba de regresión; la misma consideración de análisis se realiza en la prueba de regresión siete en la que existe una reducción de 74 casos ($119 - 45$), equivalente al 62% de reducción de la cantidad de casos de prueba..

A continuación, la Figura 3 muestra el comportamiento de este indicador en relación con el número de casos de prueba seleccionados en las diferentes pruebas de regresión y en los grupos de análisis

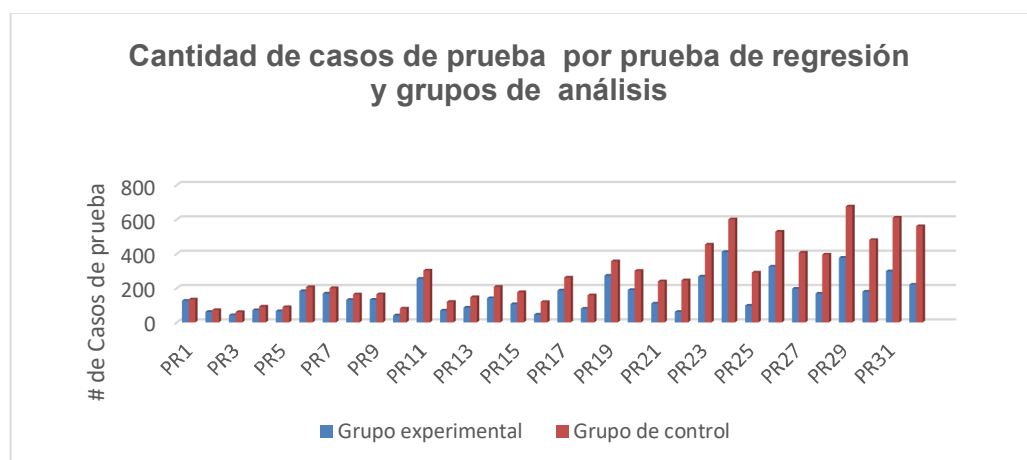


Figura 3. Cantidad de casos de prueba por prueba de regresión y grupos de análisis
Fuente: Propia

g) Indicador: Precision

La Tabla No. 37 muestra los datos recolectados para el indicador Precision de cada una de las prueba de regresión para los grupos de análisis, por ejemplo, en la prueba de regresión dos muestra la precisión de la prueba en los grupos de análisis: grupo experimental 0.09 y grupo de control 0.08; lo que indica que existe una mejora en la precisión de 0.01 de selección de casos de prueba con relación del grupo de control, en este caso, la misma consideración de análisis se realiza en la prueba de regresión siete en la que existe una precisión de 0.18 en el grupo experimental y 0.05 en el grupo de control, que equivale a decir que la precisión del método de clusterización es mayor que el método de prueba de regresión convencional.; en este caso.

A continuación, la Figura No. 4 muestra el comportamiento de este indicador en relación con el número de casos de prueba seleccionados en las diferentes pruebas de regresión y en los grupos de análisis

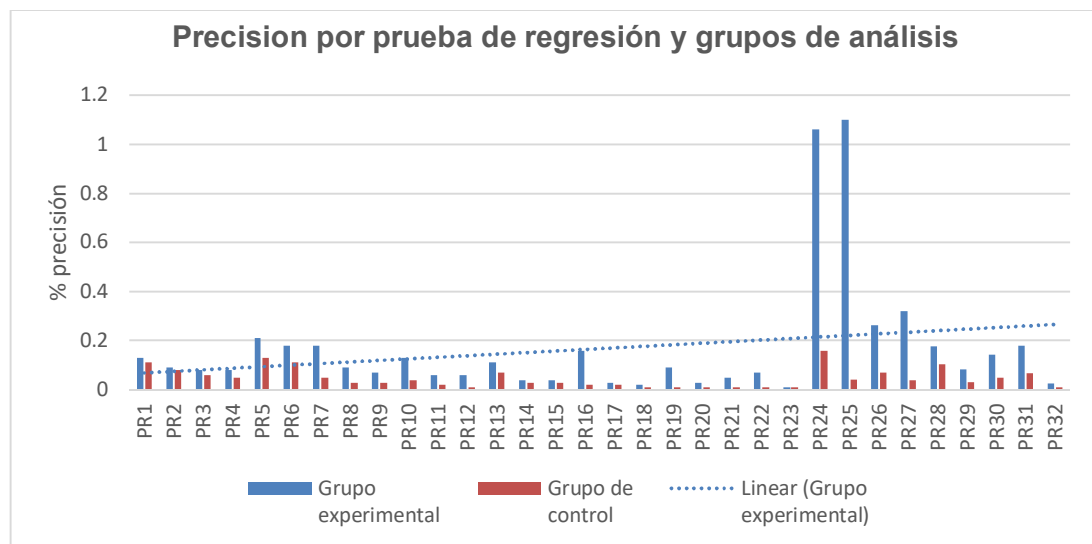


Figura 4. Precisión por prueba de regresión y grupos de análisis
Fuente: Propia

h) Indicador: Rendimiento combinado

La Tabla No. 37 muestra los datos recolectados para el indicador de Rendimiento combinado en cada una de las pruebas de regresión para los grupos de análisis, así por ejemplo, en la prueba de regresión dos se muestra el rendimiento combinado de la prueba en los grupos de análisis: grupo experimental 0.16 y grupo de control 0.15; lo que indica que existe una leve mejora en el rendimiento combinado de 0.01; la misma consideración de análisis se realiza en la prueba de regresión siete en la que existe un rendimiento combinado de 0.30 en el grupo experimental y 0.20 en el grupo de control, que equivale a decir que el rendimiento combinado del método de clusterización es mayor que el método de prueba de regresión convencional.; para las pruebas de regresión, en estos casos.

A continuación, la Figura 5 muestra el comportamiento de este indicador en relación con el rendimiento combinado en las diferentes pruebas de regresión y en los grupos de análisis.

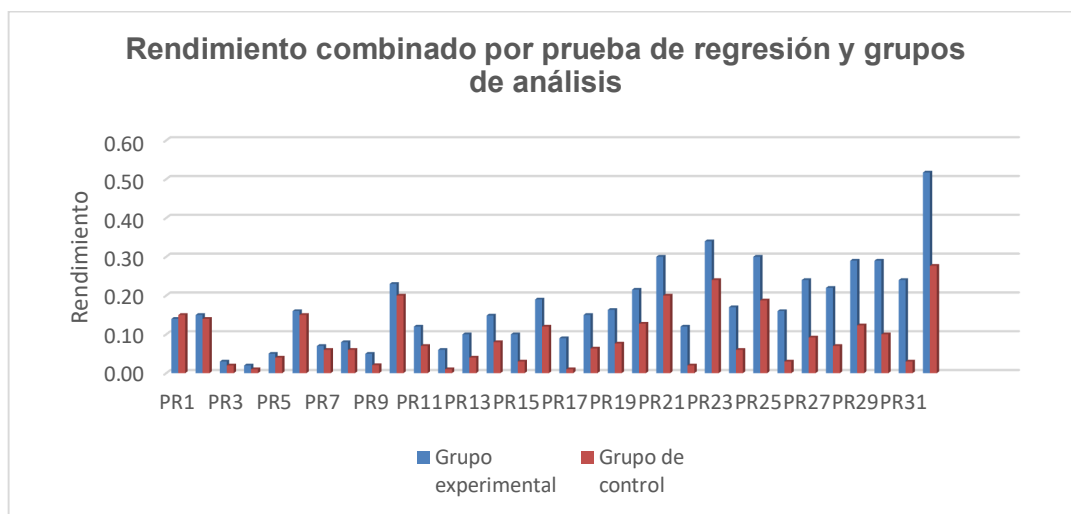


Figura 5. Rendimiento combinado por prueba de regresión y grupos de análisis
Fuente: Propia

6.4. Amenazas a la validez

Esta sección discute las amenazas a la validez que se presentan en este trabajo, para ello se siguió el estándar de clasificación de amenazas propuesto por Celeux y Govaert (1992) y Wohlin *et al.* (2012).

Amenazas internas: Son aquellas que pueden afectar la validez de los resultados desde el punto de vista de ejecución del experimento, en este caso se considera el diseño y desarrollo de los casos de prueba posterior a la construcción del producto software, así como tampoco se pudo contar con la experiencia de los desarrolladores ni tampoco poder generar patrones de diseño y desarrollo de pruebas por parte de ellos.

En lo que respecta a la clusterización, se evidencia una alta sensibilidad de los parámetros k (número de clústeres) y seed (centroide inicial) del algoritmo EM de weka, por lo que se decidió no fijar el parámetro k y dejar que el algoritmo de EM determine el mejor valor de k basado en su propio algoritmo para ello, con respecto al parámetro semilla (seed) se realizó un tratamiento de 10 semillas para reducir el efecto de la determinación del centroide, que también se relaciona al parámetro k . Existen trabajos que tratan sobre su sensibilidad (Celeux & Govaert, 1992; Hamerly & Elkan, 2002).

Amenazas externas: Dado que la investigación se realizó mediante la ejecución de caso de estudio, en la modalidad de unidades de análisis embebidas de pequeña y mediana escala, lo cual podría considerarse una limitación para generalizar los resultados, siendo necesario realizar más estudios y replicas con aplicaciones de gran escala. Sin embargo, dada las funcionalidades consideradas en cada uno de los productos software, se puede abstraer a la generalidad de las aplicaciones con características de

gestión, planeación y administración de actividades empresariales, cuyas características principales son las aplicaciones con acceso a datos.

Amenazas al constructo: Se asume que el tiempo, la severidad de los fallos y la capacidad de detección de fallos de los casos de prueba son iguales, existiendo la posibilidad de que el tiempo del fallo sea asociado a la severidad, lo cual es factible. Otro aspecto que afecta la clusterización es el patrón de defectos que deviene por el esquema relacional de la base de datos y de la experiencia de los programadores. Esta problemática aún mantiene la expectativa de la investigación sobre todo por una categorización de fallos en productos de software en general.

6.5. Contrastación de las hipótesis

En esta sección se realiza la contrastación de la hipótesis principal y las hipótesis específicas, para ello se aplicará la prueba estadística T de Student de muestras independientes de una cola a los datos de cada uno de los indicadores a las pruebas de regresión de las versiones incrementales.

Antes de la aplicación de la prueba de T, se verificará el cumplimiento del supuesto de normalidad de cada grupo de análisis en cada indicador, mediante la Kolmogorv-Smirnov con el valor de $p > 0.1$

6.5.1. Pruebas de normalidad a los datos de los grupos por indicador

A continuación, se presenta las gráficas 6, 7, 8, 9, 10 y 11 de las pruebas de normalidad aplicados a cada grupo de prueba y por indicador.

i) *Prueba de normalidad indicador: Casos de prueba*

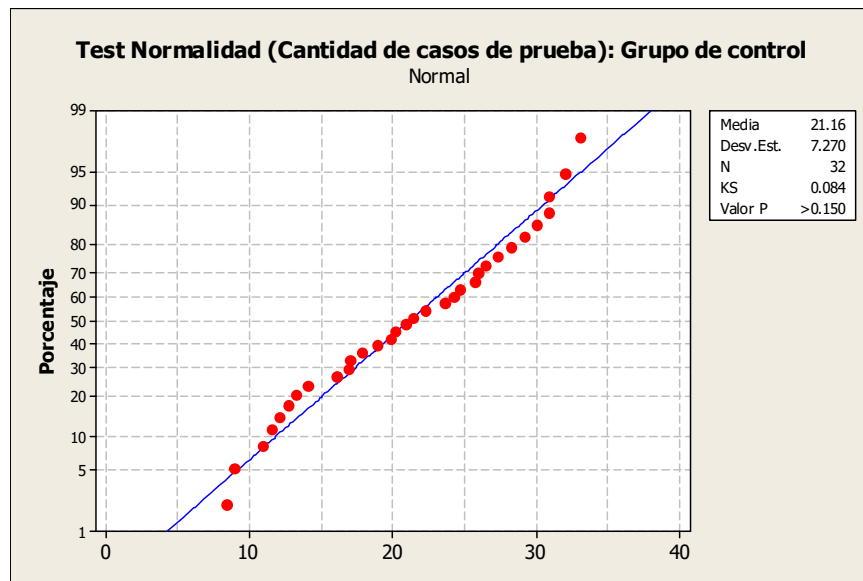


Figura 6. Test normalidad: Cantidad de casos de prueba, grupo de control

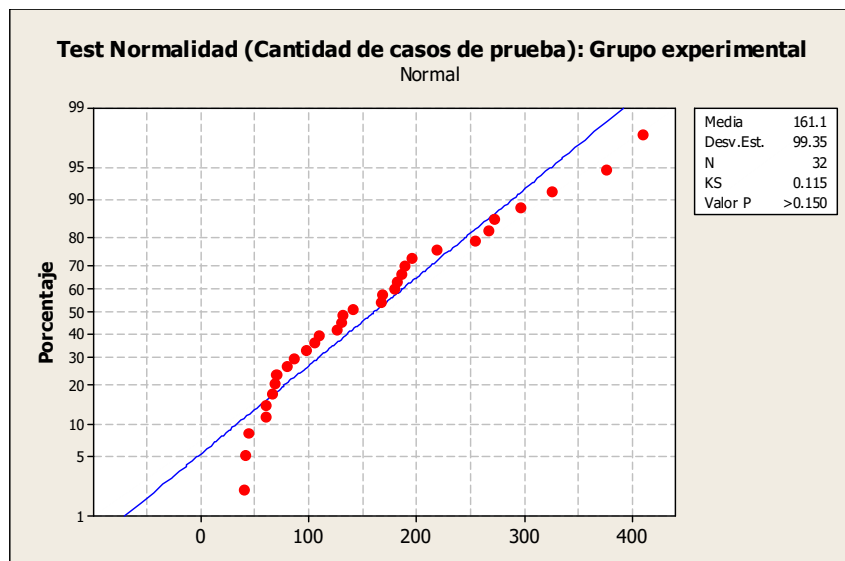


Figura 7. Test normalidad: Cantidad de casos de prueba, grupo experimental

j) *Prueba de normalidad: Precision*

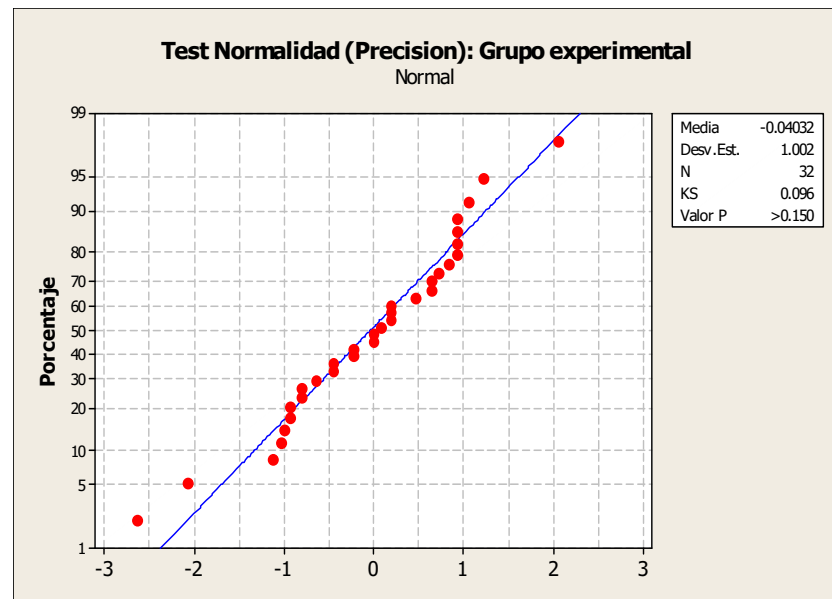


Figura 8. Test normalidad: Precision, grupo experimental

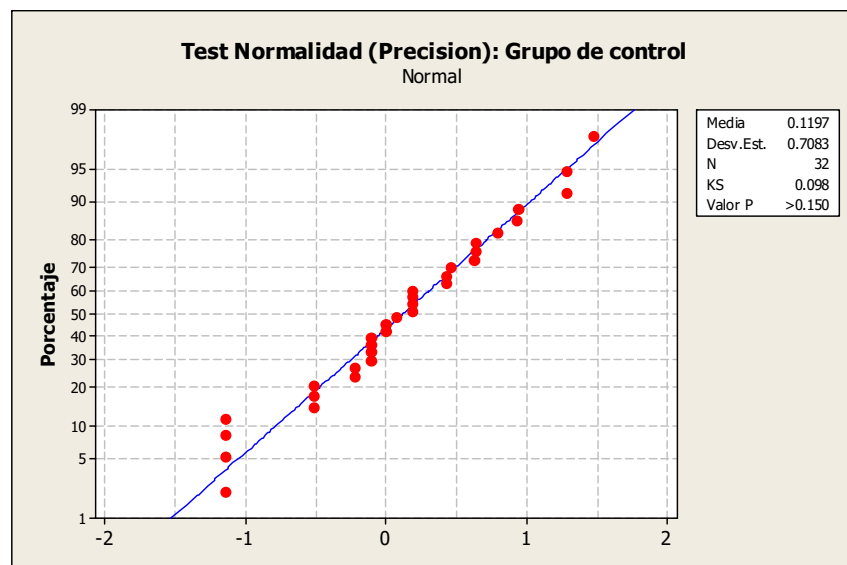


Figura 9. Test normalidad: Precision, grupo de control

k) Prueba de normalidad indicador: Rendimiento combinado

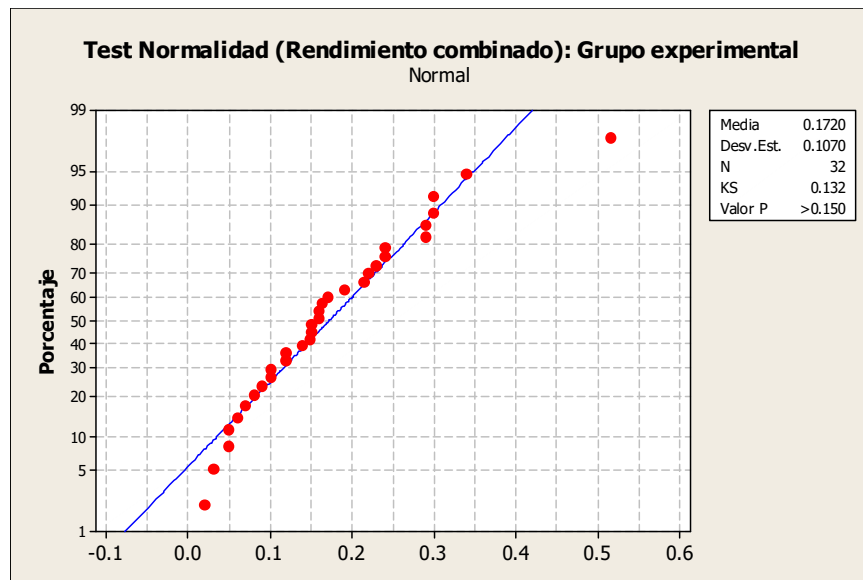


Figura 10. Test normalidad: Rendimiento combinado, grupo experimental

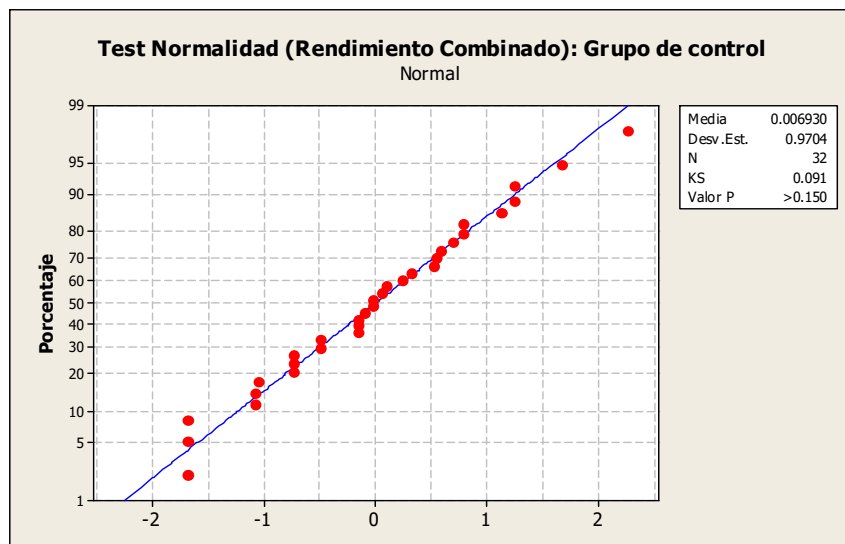


Figura 11. Test normalidad: Rendimiento combinado, grupo de control

6.5.2. Hipótesis Específica H1: Cantidad de casos de prueba

A continuación, se realiza la formulación de la hipótesis nula y alterna de H1.

H₀1: Hipótesis Nula: El método de clusterización probabilística no disminuye la cantidad de casos de prueba en pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental, que equivale a decir que sus medias se consideran diferentes y una mayor que la otra: $\mu_1 \geq \mu_2$

H₁1: Hipótesis Alterna: El método de clusterización probabilística disminuye la cantidad de casos de prueba en las pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental, que equivale a decir que $\mu_1 < \mu_2$.

a) Regla de decisión: $p \leq \alpha$: Se rechaza H₀1: caso contrario Se acepta H₁1.

Con este antecedente se realiza la prueba T de muestras independientes de una cola con un nivel de significancia alfa de 0.05, con el software estadístico Minitab, obteniéndose un valor de p de 0,0001, como se ilustra en la figura No 15.

Prueba T e IC de dos muestras: Grupo experimental; Grupo control

	N	Media	Desv.Est.	Error estándar de la media
G.exp.	32	161.1	99.4	18
G. control	32	278	178	31

Diferencia = μ (G. control.) - μ (G. exp.)

Estimado de la diferencia: 116.6

Límite superior 95% de la diferencia: 56.5

Prueba T de diferencia = 0 (vs. <): Valor T = 3.24 Valor P = 0.001 GL = 62

Ambos utilizan Desv.Est. agrupada = 144.0248

Figura 12. T Student: Cantidad de casos de prueba

- b) Resultados: En el caso de Cantidad de casos de prueba, se tiene que con $p = 0.001 < 0.05$, por lo tanto se rechaza la hipótesis nula H_0 y se acepta la hipótesis alterna H_1 , lo que sugiere que existe una reducción significativa de cantidad de casos de prueba de regresión utilizando el método de clusterización probabilístico ($m=161.1$, $es=18$, $t=-3.24$, $p < 0.05$) contra la prueba de regresión con el método de prueba de regresión convencional ($m=278$, $es=31$), con lo que se comprueba de manera formal que el método de clusterización probabilística reduce significativamente la cantidad de casos de prueba en las pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental.

La siguiente figura muestra el diagrama de caja para el indicador de Casos de prueba con el efecto de la aplicación del método de clusterización y el método convencional de la prueba de regresión en los grupos de análisis.

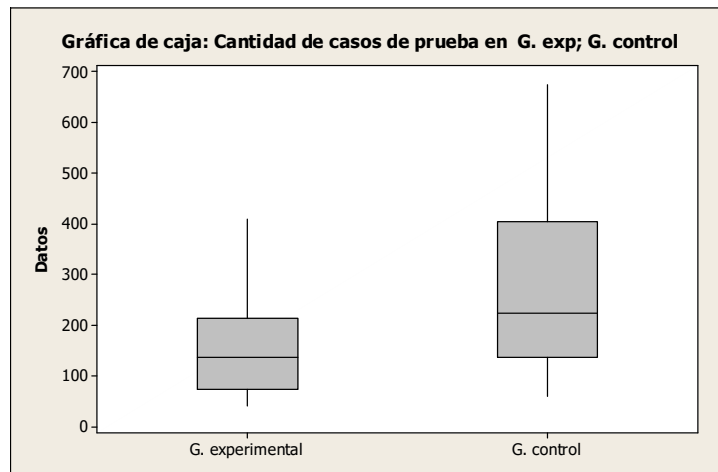


Figura 13. Cantidad de casos de prueba por grupo de análisis

Como se muestra en la figura 13, el método de clusterización probabilístico selecciona en promedio (161) promedio un subconjunto de menor tamaño de casos de prueba que el método convencional (278), por lo que el método de clusterización probabilístico reduce los casos de prueba, en promedio al método de prueba de regresión convencional con respecto a esta métrica.

6.5.3. Hipótesis Específica H2: Precision

A continuación, se realiza la formulación de la hipótesis nula y alterna de H2.

H₀2: Hipótesis Nula: El método de clusterización probabilística no mejora la precisión en pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental, que equivale a decir que sus $\mu_1 \geq \mu_2$.

H₁2: Hipótesis Alterna: El método de clusterización probabilística mejora la precisión en pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental, que equivale a decir $\mu_1 < \mu_2$.

- a) Regla de decisión: $p \leq \alpha$ Se rechaza H_0 : caso contrario se acepta H_1 .

Se precede a la aplicación de la prueba estadística inferencial de T Student con muestras independientes de una cola con un nivel de significancia alfa de 0.05, con el software minitab, obteniéndose un valor de p de 0,0001, como se ilustra en la siguiente Figura.

Prueba T e IC de dos muestras: Grupo de control; Grupo experimental

T de dos muestras para Grupo de control vs. Grupo experimental

	N	Media	Desv.Est.	Error estándar de la media
Grupo experimental	32	0.168	0.251	0.044
Grupo control	32	0.0475	0.0394	0.0070

Diferencia = μ (Grupo experimental) - μ (Grupo de control)

Estimado de la diferencia: 0.1200

Límite inferior 95% de la diferencia: 0.0452

Prueba T de diferencia = 0 (vs. >): Valor T = 2.68 Valor P = 0.0005 GL = 62

Ambos utilizan Desv.Est. agrupada = 0.1793

Figura 14. T Studente: Precisión por grupo de análisis

- b) Resultados: En el caso del indicador Precision, se tiene que con $p=0.005 < 0.05$, lo que implica que se rechaza la hipótesis nula y se acepta la hipótesis alterna, o que existe una mejora significativa con respecto a la precisión respecto del método de clusterización probabilístico ($m=0.168$, $es=0.044$ $t=2.68$, $p<0.005$) y el método de clusterización convencional ($m=0.047$, $es=0.0070$), con lo que se comprueba de manera formal que el método de clusterización probabilística mejora la precisión en pruebas de regresión en

software con acceso a datos bajo entornos de desarrollo iterativo incremental.

La siguiente figura muestra el diagrama de caja para el indicador de Precision con el efecto de la aplicación del método de clusterización y el método convencional de la prueba de regresión en los grupos de análisis

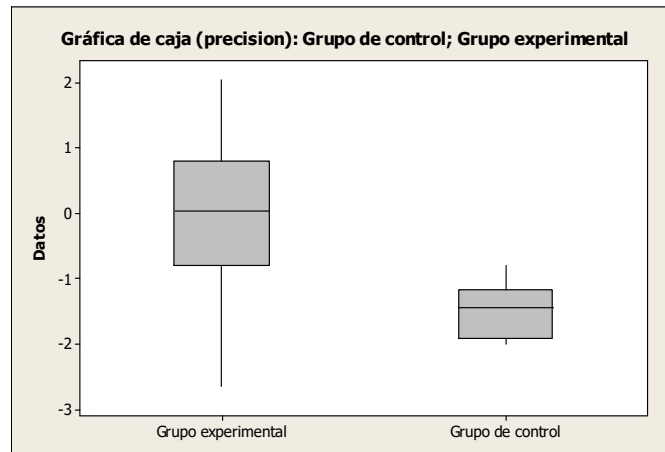


Figura 15. Precisión: Grupos de análisis

Como se muestra en la figura 15, el método de clusterización probabilístico posee una mejor precisión para las pruebas de regresión (0.17 – 17%) que el método de prueba de regresión convencional (0.05 – 5%), por lo que el método de clusterización probabilístico mejora en promedio la Precisión para una prueba de regresión con el método convencional.

6.5.4. Hipótesis Específica H3: Rendimiento combinado

A continuación, se realiza la formulación de la hipótesis nula y alterna de H3.

H₀3: Hipótesis Nula: El método de clusterización probabilística no mejora el el rendimiento combinado de las pruebas de regresión de software

con acceso a datos bajo entornos de desarrollo iterativo incremental, que equivale a decir que sus $\mu_1 \geq \mu_2$.

H₁₃: Hipótesis Alternativa: El método de clusterización probabilística mejora el rendimiento combinado de las pruebas de regresión de software con acceso a datos bajo entornos de desarrollo iterativo incremental, que equivale a decir $\mu_1 < \mu_2$.

a) Regla de decisión: $p \leq \alpha$, Rechazo del H₀₃: caso contrario se acepta H₁₃.

Con este antecedente se realiza la prueba de T Student, con muestras independientes de una cola, con un nivel de significancia alfa de 0.05, con apoyo de la herramienta Minitab, obteniéndose un valor de p de 0,0001, como se ilustra en la Figura.

Prueba T e IC de dos muestras: Grupo de control vs Grupo experimental

T de dos muestras para Grupo de control vs. Grupo experimental

	N	Media	Desv.Est.	Error estándar de la media
Grupo de control	32	0.0909	0.0715	0.013
Grupo de experimental	32	0.172	0.107	0.019

Diferencia = μ (Grupo experimental) - μ (Grupo control)

Estimado de la diferencia: 0.0811

Límite superior 95% de la diferencia: 0.0432

Prueba T de diferencia = 0 (vs. <): Valor T = 3.57 Valor P = 0.0001 GL = 62

Ambos utilizan Desv.Est. agrupada = 0.0910

Figura 16. T student: Rendimiento combinado

b) Resultados: En el caso de rendimiento combinado, se tiene que con $p = 0.0001 < 0.05$, por lo que se rechaza H₀₃ y se acepta H₁₃, lo que implica que existe una mejora significativa respecto del método de clusterización probabilístico ($m=0.172$, $es=0.019$ $t=3.57$ $p<0.05$) y el método convencional en las pruebas de regresión ($m=0.09$,

es=0.013), con lo que se comprueba de manera formal que el método de clusterización probabilística mejora el rendimiento combinado de las pruebas de regresión de software con acceso a datos bajo entornos de desarrollo iterativo incremental.

La siguiente figura muestra el diagrama de caja para el indicador de Rendimiento combinado con el efecto de la aplicación del método de clusterización probabilístico y el método convencional para las pruebas de regresión en los grupos de análisis.

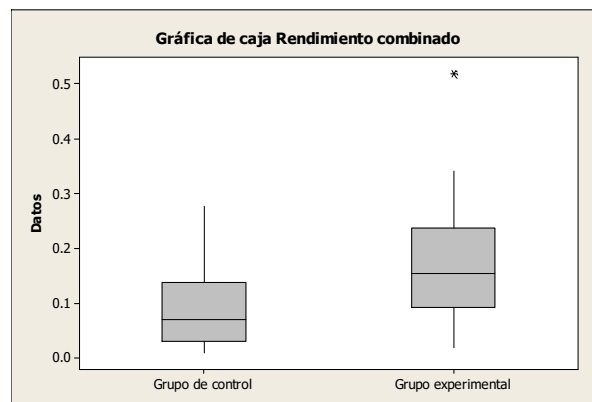


Figura 17. Rendimiento combinado por grupos de análisis

Como se muestra en la Figura 17, el método de clusterización probabilístico mejora el rendimiento combinado en promedio (0.17) al de la Rendimiento combinado del método convencional en las pruebas de regresión de software basado en acceso a los datos en entorno de desarrollo iterativo incremental.

A continuación, se presenta la Tabla 38 con el resumen del comportamiento de los indicadores en las pruebas de regresión utilizando el método de clusterización probabilístico y el método convencional en los grupos de experimentación y de control respectivamente.

Tabla 38. Comportamiento de los indicadores por grupos de análisis

Indicador	Grupo experimental	Grupo de control
Cantidad de casos de prueba	161 (58%)	278 (100%)
Precision	0.17 (17%)	0,05 (5%)
Rendimiento combinado	0.17 (17%)	0.09 (9%)

6.5.5. Hipótesis general

Se procede a la comprobación de la hipótesis general:

H_0G : El método de clusterización probabilística no mejora la eficiencia de las pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental.

H_1G : En método de clusterización probabilística mejora las pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental.

Tabla 39. Valores de p de T Student en la contrastación de hipótesis

Indicador	Valor p
Cantidad de casos de prueba	0.001
Precisión	0.0005
Rendimiento combinado	0.0001

Con los valores de los indicadores de la Tabla 38 y los valores de la inferencia estadística de p que se muestra en la Tabla 39 se evidencia la mejora de la prueba de regresión de software en los grupos de análisis, de manera específica, hay una reducción de la cantidad de casos de prueba para realizar la prueba de regresión del 116 que equivale a un 42%, con una capacidad de detección de fallas del 100%, con una precisión del 15%, y con un rendimiento cambiando del 0,17 que equivale a un 17% para el rendimiento global del método, frente a una reducción de 0 casos de prueba de 278, que

equivale a un 1000%, con una precisión de 5% y un rendimiento combinado de 9%.

Estadísticamente, con el valor de $p < 0,05$ de la prueba estadística inferencial nos permite rechazar la hipótesis nula H_0G , y aceptar como válida la hipótesis alterna H_1G : El método de clusterización probabilística mejora las pruebas de regresión en software con acceso a datos bajo entornos de desarrollo iterativo incremental

6.6. Reporte

En esta sección se realiza la presentación de los diferentes hallazgos a través del procedimiento de análisis de datos, según la estructura de Runeson y Zelkowitz (2005), se inicia con la sección de objetivo.

- **Objetivo:** El objetivo es analizar el comportamiento de dos métodos de pruebas de regresión de software, para ello se ha considerado los indicadores de Cantidad de casos de prueba, la precisión y el rendimiento combinado.

Los métodos en cuestión son el método de clusterización probabilístico propuesto y el método convencional para pruebas de regresión de software, considerando como método de control al método convencional de regresión total.

los indicadores considerados, por ejemplo, cantidad de casos de prueba, capacidad de detección, precisión y el rendimiento combinado son tomados desde el punto de vista de la teoría de la información y en su enfoque en el comportamiento en la capacidad de búsqueda y recuperación de información.

Las cuestiones de investigación se representaron como hipótesis por cada uno de los indicadores consideradas, por lo que se verificó el comportamiento comparativo de las dos metodos a través de las hipótesis planteadas.

- **Sujetos de estudio:** el sujeto de estudio está conformado por las 32 pruebas de regresión de cada una de las versiones de cinco productos software de tamaño (pequeño y medio) con funcionalidades similares (ámbito

de SI, CRM y ERP) y que fueron desarrolladas en un ambiente iterativo incremental en un entorno de desarrollo Java, casos de prueba unitarios y el esquema de la base de datos relacional.

El número de versiones de cada producto es diferente: Estafeta 5 versiones, Silabo 9 versiones, Académico con 12 versiones, GICO 5 y YELLOW 6, dando un total de 37 versiones en total, a partir de la segunda versión se registró el número de fallos en cada una de ellas, datos que se han presentado en las tablas correspondientes.

- **Análisis de datos:** La estrategia de análisis ha sido aplicar un estudio por indicador (cantidad de casos de prueba, precisión, y rendimiento combinado), por método y grupo de análisis.

Por lo anterior, se cuenta con resultados de los análisis por indicador: método y grupo de analisis. El objetivo es observar el comportamiento de los métodos en los grupos de análisis.

El análisis estadístico considerado fue la prueba T Student de muestras independientes de una sola cola con cada uno de los indicadores.

Como las variables de estudio son de carácter cualitativo, se consideró el intervalo de confianza del 95%, con un valor de significancia del 5% en cada una de las pruebas T realizadas.

Las cuestiones que guiaron este trabajo se las consideró como hipótesis, siendo tres hipótesis para considerar según la matriz de consistencia presentada en el capítulo de metodología.

- **Resultados:** Para este paso se considera los datos de la Tabla 40 a continuación se procede a analizar por filas y por columnas.

Tabla 40. Comportamiento de los indicadores por grupos de analisis

Indicador	Grupo experimental	Grupo de control
Cantidad de casos de prueba	161 (58%)	278 (1005)
Precision	0.17) (17%)	0.05 (5%)
Rendimiento combinado	0.17 (17%)	0.09 (9%)

Por columnas se observa el comportamiento de los metodos en cada grupo por cada indicador, se presenta el valor p para la decisión de aceptación o rechazo de las hipótesis, además, notándose que se rechaza las hipótesis nulas y se acepta las hipótesis alternas en cada una de ellas

Como resultado final general se determina que el metodo propuesto tiene un mejor comportamiento, respecto de los indicadores considerados, que el método convencional evidenciado en los grupos de análisis, considerando el enfoque de indicadores de búsqueda y recuperación de información: cantidad de casos de prueba, precisión y rendimiento combinado en pruebas de regresión de software con acceso a datos bajo el entorno de desarrollo iterativo incremental.

CAPÍTULO VII: CONCLUSIONES Y TRABAJOS FUTUROS

7.1. Conclusiones

La revisión sistemática de la literatura realizada y presentada en este trabajo identificó los principales aspectos involucrados en las pruebas de regresión de software, centrándose en factores como la etapa de desarrollo o de mantenimiento (progresivo, correctivo), notando que mayoritariamente se realizan en contexto de mantenimiento y con productos de software de corte académico-didáctico (software de programación de sistemas); por otro lado, fue de interés conocer los métodos/técnicas, métricas e indicadores asociados a este tipo de prueba, siendo de interés de esta investigación, la cantidad de casos de prueba, la precisión, la recuperación (capacidad de detección de fallas) y el rendimiento combinado. Finalmente se determinó que el tamaño de producto de software en los cuales se analizaba las pruebas de regresión es de pequeña escala (menos de 10K).

En tal contexto, el reto general se orientó al diseño de un método que considere la mejora de los indicadores: cantidad de casos de prueba, así como las del paradigma de búsqueda y recuperación de la información como precisión (precisión en seleccionar los casos que detectan fallos), recuperación (capacidad de detección de fallos de los casos de prueba), y el de rendimiento combinado.

Para el aspecto de acceso a datos, se consideró la selección de los casos de prueba que realizan accesos a los datos, realizando un análisis

textual del conjunto de casos de prueba del producto software que contengan operaciones de lectura, escritura, modificación o eliminación de datos, una vez realizado lo anterior se procede con un proceso de filtración con el esquema de bases de datos para determinar qué campos de la base de datos son los considerados en aquellas operaciones. A continuación, mediante una técnica de minería de datos (clusterización probabilística), se determina los clústeres de casos de prueba que realizan el mismo tipo de acceso a datos en un determinado campo, con la idea fundamental de agrupar los casos de prueba, que dado el caso de que fallen, estén en un mismo clúster y ejecutarlos en una prueba de regresión.

El proceso de clusterización, en general, adolece de dos debilidades que fueron minimizadas en este trabajo: el primero, la determinación del número de clústeres a formar (K), y el segundo, la determinación aleatoria del centroide inicial en su algoritmo; para la primera, se seleccionó el método de clusterización no supervisada denominada Expectation Maximization (EM) que permite la determinación automática del número de clústeres en función del análisis probabilístico de los datos; para la segunda, se consideró la estrategia de realizar una serie de ejecuciones del proceso de clusterización de una misma versión para estadísticamente minimizar sesgos en los resultados y su impacto aleatorio.

Para validar la propuesta, se realizó un estudio experimental bajo el paradigma de la ingeniería de software experimental, con 32 pruebas de 37 regresión provenientes cinco productos software con acceso a datos y desarrollados en entorno iterativo incremental de pequeña y mediana escala; para comparar el comportamiento del método propuesto se contrastó con el método convencional de prueba de regresión (regresión total). El análisis estadístico formal fue realizado mediante prueba de T de Student para muestras independiente de una cola, con un nivel de significancia del 5%, con una probabilidad del 95%, con el cumplimiento del supuesto de normalidad con la prueba Kolmogorv Smirnov.

Los resultados del trabajo sugieren hallazgos importantes:

En el caso del indicador de Cantidad de casos de prueba, en promedio, el método propuesto selecciona 161 (58%) casos de prueba frente a 278 (100%) del método convencional, que, vistos en porcentaje, el método propuesto selecciona un 58% de casos de prueba mientras que el método convencional siempre selecciona el 100% de los casos de prueba. Este análisis sugiere una reducción de 117 casos de prueba para una prueba de regresión en software con accesos a datos en entornos de desarrollo iterativo incremental (42%).

Con respecto al indicador Precision del método propuesto frente al método convencional, sugiere una mejora en la precisión del método propuesto del 17% frente al 5% del método convencional de pruebas de regresión.

En este trabajo, por diseño del método seguro, el indicador Recuperación (capacidad de detección de fallos) es del 100%, ya que siempre considera todos los fallos de la versión.

Finalmente, en el análisis del indicador de Rendimiento combinado se evidencia una mejora del 0,17 del método propuesto frente al del 0.09 del método convencional, que, vistos en porcentaje es una mejora global del proceso del 17% frente al 9% respectivamente.

De manera general, después de la realización del trabajo, la evidencia sugiere que el método de clusterización probabilístico propuesto mejora las pruebas de regresión frente al método convencional, otorgando de esta

manera mayor confianza en los desarrolladores en el proceso, así como en los usuarios de contar con productos de software de mejor confiabilidad.

7.2. Trabajos futuros

De manera general, los proyectos futuros son la continuación del presente, mejorando la propuesta. En esa dirección se considera los siguientes trabajos relacionados:

1. Análisis, clasificación y tipología de fallos con los que se puede mejorar el proceso de clusterización probabilístico.
2. Investigar sobre el tratamiento de casos de prueba con detección de distintos tipos de fallos y de número de fallos.
3. Complementar con herramientas a manera de plugins de tal manera que sea automatizada la tarea de una prueba de regresión, enfocada en el proceso iterativo incremental.
4. Investigar y complementar la propuesta para un servicio de pruebas de regresión de software en la nube.
5. Aplicación de varios frameworks y lenguajes. JUnit es un framework de testing ampliamente utilizado en el entorno de Java, pero se podría investigar la aplicabilidad de otros frameworks, como por ejemplo TestNG, Selenium WebDriver.
6. Proyecto de automatización de la gestión y seguimiento de fallos para pruebas de regresión.

BIBLIOGRAFÍA

A Comparative Study of Clustering Deviations for Black-Box Regression

Test-Selection Techniques—ProQuest. (s. f.). Recuperado 6 de agosto de 2017, de

<https://search.proquest.com/openview/4530a4c0e8e88836f02e252430f16217/1?pq-origsite=gscholar&cbl=1606379>

Acebo, M., & Nuñez, A. (2016). Estudios Industriales: Orientación estratégica para la toma de decisiones – Industria de Software | ESPAE Graduate School of Management - ESPOL. Acebo, M., & Nuñez, A. (2016).

Estudios Industriales Orientacion Estrategica para la toma de decisiones. <http://www.espae.espol.edu.ec/industria-software/>

Agile Manifesto for Software Development. (2015, junio 29). *Agile Alliance.*

<https://www.agilealliance.org/agile101/the-agile-manifesto/>

Ajayi, A. I., Omirin, & F, F. (2007). The use of management information systems (MIS) in decision making in the South-West Nigerian Universities. *Educational Research and Reviews*, 2(5), 109-116.

Ammann, P. E., & Knight, J. C. (1988). Data diversity: An approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4), 418-425. <https://doi.org/10.1109/12.2185>

Amo, F. A., Normand, L. M., & Pérez, F. J. S. (2005). *Introducción a la ingeniería del software*. Delta Publicaciones.

Anderson, J., Salem, S., & Do, H. (2014a). Improving the Effectiveness of Test Suite Through Mining Historical Data. *Proceedings of the 11th*

Working Conference on Mining Software Repositories, 142-151.

<https://doi.org/10.1145/2597073.2597084>

Anderson, J., Salem, S., & Do, H. (2014b). Improving the Effectiveness of Test Suite Through Mining Historical Data. *Proceedings of the 11th Working Conference on Mining Software Repositories*, 142-151.

<https://doi.org/10.1145/2597073.2597084>

Arora, P. K., & Bhatia, R. (2018). A Systematic Review of Agent-Based Test Case Generation for Regression Testing. *Arabian Journal for Science and Engineering*, 43(2), 447-470. <https://doi.org/10.1007/s13369-017-2796-4>

Ati, P., & Fabricio, E. (2017). *Desarrollo de pruebas unitarias automáticas para el sistema web escolar del Instituto Tecnológico Superior Stanford*. <http://dspace.esPOCH.edu.ec/handle/123456789/7459>

Ball, T. (1998). On the Limit of Control Flow Analysis for Regression Test Selection. *Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis*, 134-142.

<https://doi.org/10.1145/271771.271802>

Beizer, B. (1990). *Software Testing Techniques (2Nd Ed.)*. Van Nostrand Reinhold Co.

Bentley L. D., Dittman, K. C., & Whitten. (2000). *Systems analysis and Design Methods*. Irwin /McGraw Hill.

Berkhin, P. (2006). A Survey of Clustering Data Mining Techniques. En J. Kogan, C. Nicholas, & M. Teboulle (Eds.), *Grouping Multidimensional Data* (pp. 25-71). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-28349-8_2

- Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. *2007 Future of Software Engineering*, 85-103.
<https://doi.org/10.1109/FOSE.2007.25>
- Bilenko, M., Basu, S., & Mooney, R. J. (2004). Integrating Constraints and Metric Learning in Semi-supervised Clustering. *Proceedings of the Twenty-first International Conference on Machine Learning*, 11-.
<https://doi.org/10.1145/1015330.1015360>
- Biswas, S., Mall, R., & Satpathy, M. (2013). A Regression Test Selection Technique for Embedded Software. *ACM Trans. Embed. Comput. Syst.*, 13(3), 47:1-47:39. <https://doi.org/10.1145/2539036.2539043>
- Biswas, S., Mall, R., Satpathy, M., & Sukumaran, S. (2009). A Model-based Regression Test Selection Approach for Embedded Applications. *SIGSOFT Softw. Eng. Notes*, 34(4), 1-9.
<https://doi.org/10.1145/1543405.1543413>
- Broncano, M., & Alexander, M. (2016). *Sistema de seguimiento a planificación de PEA para la Facultad de Informática y Electrónica de la ESPOCH*. <http://dspace.esPOCH.edu.ec/handle/123456789/6329>
- Brooking, A. G. (1984). Towards a Methodology for a Design of Knowledge Based Systems. *Proceedings of the 1984 Annual Conference of the ACM on The Fifth Generation Challenge*, 4-.
<https://doi.org/10.1145/800171.809595>
- Cartaxo, E. G., Machado, P. D. L., & Neto, F. G. O. (2011). On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability*, 21(2), 75-100.
<https://doi.org/10.1002/stvr.413>

- Celeux, G., & Govaert, G. (1992). A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics & Data Analysis*, 14(3), 315-332. [https://doi.org/10.1016/0167-9473\(92\)90042-E](https://doi.org/10.1016/0167-9473(92)90042-E)
- Chays, D., Deng, Y., Frankl, P. G., Dan, S., Vokolos, F. I., & Weyuker, E. J. (2004). An AGENDA for testing relational database applications. *Software Testing, Verification and Reliability*, 14(1), 17-44. <https://doi.org/10.1002/stvr.286>
- Chen, S., Chen, Z., Zhao, Z., Xu, B., & Feng, Y. (2011). Using semi-supervised clustering to improve regression test selection techniques. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST)*, 1-10. <https://doi.org/10.1109/ICST.2011.38>
- Chen, Y.-F., Rosenblum, D. S., & Vo, K.-P. (1994). TestTube: A System for Selective Regression Testing. *Proceedings of the 16th International Conference on Software Engineering*, 211-220. <http://dl.acm.org/citation.cfm?id=257734.257769>
- Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011). Using program slicing to improve the efficiency and effectiveness of cluster test selection. *International Journal of Software Engineering and Knowledge Engineering*, 21(06), 759-777. <https://doi.org/10.1142/S0218194011005487>
- Cibulski, H., & Yehudai, A. (2011). Regression Test Selection Techniques for Test-Driven Development. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 115-124. <https://doi.org/10.1109/ICSTW.2011.28>

- D. Marijan, M. Liaaen, A. Gotlieb, S. Sen and C. Ieva. (2017). TITAN: Test Suite Optimization for Highly Configurable Software. *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 524-531. <https://doi.org/10.1109/ICST.2017.60>
- Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *Systems Analysis and Design: An Object-Oriented Approach with UML*. John Wiley & Sons.
- Di Nardo, D., Alshahwan, N., Briand, L., & Labiche, Y. (2015). Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. *Software Testing, Verification and Reliability*, 25(4), 371-396. <https://doi.org/10.1002/stvr.1572>
- Dickinson, W., Leon, D., & Podgurski, A. (2001). Finding Failures by Cluster Analysis of Execution Profiles. *Proceedings of the 23rd International Conference on Software Engineering*, 339-348.
<http://dl.acm.org/citation.cfm?id=381473.381509>
- Do, H., Rothermel, G., & Kinneer, A. (2006). Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis. *Empirical Software Engineering*, 11(1), 33-70. <https://doi.org/10.1007/s10664-006-5965-8>
- Duarte, A., Cirne, W., Brasileiro, F., & Machado, P. (2006). GridUnit: Software Testing on the Grid. *Proceedings of the 28th International Conference on Software Engineering*, 779-782.
<https://doi.org/10.1145/1134285.1134410>
- Engström, E., Runeson, P., & Skoglund, M. (2010). A Systematic Review on Regression Test Selection Techniques. *Inf. Softw. Technol.*, 52(1), 14-30. <https://doi.org/10.1016/j.infsof.2009.07.001>

- Ficco, M., Pietrantuono, R., & Russo, S. (2011). Bug Localization in Test-driven Development. *Adv. Soft. Eng.*, 2011, 2:1-2:18.
<https://doi.org/10.1155/2011/492757>
- Gamma, E., & Beck, K. (2000). *{JUnit: A Regression Testing Framework}*.
- Garlan, D. (2014). Software Architecture: A Travelogue. *Proceedings of the on Future of Software Engineering*, 29-39.
<https://doi.org/10.1145/2593882.2593886>
- Gladys Machado Pereira S. Lima, Arilo C. Dias Neto, & Guilherme H. Travassos,. (2008). Heuristics for the Identification of Class Integration Order in Object-Oriented Software Testing. *CLEI electronic 2006*, 11, 13.
- Gökçe, N., Belli, F., Eminli, M., & Dinçer, B. T. (2015). Model-based test case prioritization using cluster analysis: A soft-computing approach. *Turkish Journal of Electrical Engineering and Computer Science*, 23(3), 623.
- Gómez, O. S., Batún, J. L., & Aguilar, R. A. (2013). Pair versus Solo Programming—An Experience Report from a Course on Design of Experiments in Software Engineering. *arXiv:1306.4245 [cs]*.
<http://arxiv.org/abs/1306.4245>
- Gómez, O. S., Juristo, N., & Vegas, S. (2014). Understanding replication of experiments in software engineering: A classification. *Information and Software Technology*, 56(8), 1033-1048.
<https://doi.org/10.1016/j.infsof.2014.04.004>
- Gorton, I. (2006). *Essential Software Architecture*. Springer Science & Business Media.

- Guamán, Y., Rocío, G., Zabala, M., & Cecibel, T. (2016). *Desarrollo de una Aplicación Web para el Registro de la Distribución de la Jornada Laboral del Personal Académico de la Facultad de Informática y Electrónica*. <http://dspace.esPOCH.edu.ec/handle/123456789/4761>
- Guo, B., Subramaniam, M., & Chundi, P. (2012a). Analysis of Test Clusters for Regression Testing. *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, 736-736. <https://doi.org/10.1109/ICST.2012.165>
- Guo, B., Subramaniam, M., & Chundi, P. (2012b). *Analysis of Test Clusters for Regression Testing*. 736-736. <https://doi.org/10.1109/ICST.2012.165>
- Guzmán, J. C., López, G., & Pacheco, A. (2017). Defining 'Architecture' for Software Engineering – A Review of Terminology. *Advances in Human Factors, Software, and Systems Engineering*, 143-152. https://doi.org/10.1007/978-3-319-60011-6_14
- Haftmann, F., Kossmann, D., & Lo, E. (2007). A framework for efficient regression tests on database applications. *The VLDB Journal*, 16(1), 145-164. <https://doi.org/10.1007/s00778-006-0028-8>
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1), 10-18. <https://doi.org/10.1145/1656274.1656278>
- Hamerly, G., & Elkan, C. (2002). Alternatives to the K-means Algorithm That Find Better Clusterings. *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, 600-607. <https://doi.org/10.1145/584792.584890>

- Haraty, R. A., Mansour, N., & Daou, B. (2001). Regression Testing of Database Applications. *Proceedings of the 2001 ACM Symposium on Applied Computing*, 285-289. <https://doi.org/10.1145/372202.372342>
- Haraty, R. A., Mansour, N., Moukahal, L., & Khalil, I. (2016). Regression Test Cases Prioritization Using Clustering and Code Change Relevance. *International Journal of Software Engineering and Knowledge Engineering*, 26(05), 733-768. <https://doi.org/10.1142/S0218194016500248>
- Harrold, M. J. (1999). Testing evolving software¹This work was supported in part by grants from Microsoft Inc. And Boeing Airplane Group, and by NSF under NYI Award CCR-9696157 and ESS Award CCR-9707792 to Ohio State University.1. *Journal of Systems and Software*, 47(2), 173-181. [https://doi.org/10.1016/S0164-1212\(99\)00037-0](https://doi.org/10.1016/S0164-1212(99)00037-0)
- Hetzel, W. C., & Hetzel, B. (1991). *The Complete Guide to Software Testing* (2nd ed.). John Wiley & Sons, Inc.
- Hummel, M. (2014). State-of-the-Art: A Systematic Literature Review on Agile Information Systems Development. *2014 47th Hawaii International Conference on System Sciences*, 4712-4721. <https://doi.org/10.1109/HICSS.2014.579>
- Jon Whittle, John Hutchinson, Mark Rouncefield. (2014). The state of practice in model-driven engineering. *IEEE Software*, 3(31), 79-85. <https://doi.org/10.1109/MS.2013.65>
- Jung, Y. G., Kang, M. S., & Heo, J. (2014). Clustering performance comparison using K-means and expectation maximization algorithms.

Biotechnology & Biotechnological Equipment, 28(sup1), S44-S48.

<https://doi.org/10.1080/13102818.2014.949045>

Juristo, N., Moreno, A. M., Vegas, S., & Solari, M. (2006). In Search of What We Experimentally Know about Unit Testing. *IEEE Software*, 23(6), 72-80. <https://doi.org/10.1109/MS.2006.166>

Juristo, Natalia, & Moreno, A. M. (2013). *Basics of Software Engineering Experimentation*. Springer Science & Business Media.

Kazmi, R., Jawawi, D. N. A., Mohamad, R., & Ghani, I. (2017). Effective Regression Test Case Selection: A Systematic Literature Review. *ACM Comput. Surv.*, 50(2), 29:1-29:32. <https://doi.org/10.1145/3057269>

Kim, S., Zimmermann, T., Whitehead Jr., E. J., & Zeller, A. (2007). Predicting Faults from Cached History. *Proceedings of the 29th International Conference on Software Engineering*, 489-498. <https://doi.org/10.1109/ICSE.2007.66>

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., & Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8), 721-734. <https://doi.org/10.1109/TSE.2002.1027796>

Kitchenham, B., Pickard, L., & Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE Software*, 12(4), 52-62. <https://doi.org/10.1109/52.391832>

Kitchenham, Barbara, Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software

engineering – A systematic literature review. *Information and Software Technology*, 51(1), 7-15. <https://doi.org/10.1016/j.infsof.2008.09.009>

Kumar, M., Sharma, A., & Kumar, R. (2012). Multi faceted measurement framework for test case classification and fitness evaluation using fuzzy logic based approach. *Chiang Mai Journal of Science*, 39(3).

Kumar, M., Sharma, A., & Kumar, R. (2014). Fuzzy entropy-based framework for multi-faceted test case classification and selection: An empirical study. *IET Software*, 8(3), 103-112. <https://doi.org/10.1049/iet-sen.2012.0198>

Kumar, P., Jandsalar, H., & Rawat, M. (2017). Review of Information Retrieval models. *International Journal of Research and Engineering*, 4(4), 98-100.

Kumari, A. C., Srinavas, K., & Gupta, M. P. (2013). RegressAid–A CASE Tool for Minimization of Test Suite for Regression Testing. *International Journal of Computer Applications*, 71(18).

Kung, S. Y. (2014). *Kernel Methods and Machine Learning*. Cambridge University Press.

Laplante, P. A. (2007). *What Every Engineer Should Know about Software Engineering*. CRC Press.

Leung, H. K. N., & White, L. (1991). A cost model to compare regression test strategies. *Conference on Software Maintenance, 1991., Proceedings*, 201-208. <https://doi.org/10.1109/ICSM.1991.160330>

Leung, H. K. N., & White, L. (1990). A study of integration testing and software regression at the integration level. *Proceedings. Conference*

on *Software Maintenance 1990*, 290-301.

<https://doi.org/10.1109/ICSM.1990.131377>

Leung, H. K. N., & White, L. (1989). Insights into regression testing [software testing]. *Proceedings. Conference on Software Maintenance - 1989*, 60-69. <https://doi.org/10.1109/ICSM.1989.65194>

Lonetti, F., & Marchetti, E. (2018). Emerging Software Testing Technologies. En *Advances in Computers*. Elsevier. <https://doi.org/10.1016/bs.adcom.2017.11.003>

Maia, C. L. B., do Carmo, R. A. F., de Freitas, F., Gomes, C., de Campos, G. A. L., & de Souza, J. T. (2010). Automated Test Case Prioritization with Reactive GRASP. *Advances in Software Engineering, 2010*, e428521. <https://doi.org/10.1155/2010/428521>

Maia, C. L. B., do Carmo, R. A. F., de Freitas, F., Gomes, C., de Campos, G. A. L., de Souza, J. T., Maia, C. L. B., do Carmo, R. A. F., de Freitas, F., Gomes, C., de Campos, G. A. L., & de Souza, J. T. (2010). Automated Test Case Prioritization with Reactive GRASP, Automated Test Case Prioritization with Reactive GRASP. *Advances in Software Engineering, Advances in Software Engineering, 2010, 2010*. <https://doi.org/10.1155/2010/428521>, [10.1155/2010/428521](https://doi.org/10.1155/2010/428521)

Matalonga, S., & Solari, M. (2013). *Un programa de investigación en deuda técnica de software*. <https://dspace.ort.edu.uy/handle/20.500.11968/2724>

Mirarab, S., Akhlaghi, S., & Tahvildari, L. (2012). Size-Constrained Regression Test Case Selection Using Multicriteria Optimization. *IEEE*

Transactions on Software Engineering, 38(4), 936-956.

<https://doi.org/10.1109/TSE.2011.56>

Mohanty, R., Ravi, V., & Patra, M. R. (2010). The application of intelligent and soft-computing techniques to software engineering problems: A review. *International Journal of Information and Decision Sciences*, 2(3), 233-272. <https://doi.org/10.1504/IJIDS.2010.03345>

Muşlu, K., Brun, Y., Ernst, M. D., & Notkin, D. (2013). Making Offline Analyses Continuous. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 323-333. <https://doi.org/10.1145/2491411.2491460>

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing* (3rd ed.). Wiley Publishing.

O'Brien, J. A., & George Marakas. (2010). *Management Information Systems* (10 edition). McGraw-Hill Education.

Orso, A., Shi, N., & Harrold, M. J. (2004). Scaling Regression Testing to Large Software Systems. *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, 241-251. <https://doi.org/10.1145/1029894.1029928>

Özkan, R., Garousi, V., & Betin-Can, A. (2017). *Multi-objective regression test selection in practice: An empirical study in the defense software industry*. <http://orbilu.uni.lu/handle/10993/31852>

Panigrahi, C. R., & Mall, R. (s. f.). A Hybrid Regression Test Selection Technique for Object-Oriented Programs. *International Journal of Software Engineering and Its Applications*, 6(4), 17-34.

- Panigrahi, C. R., & Mall, R. (2010). Model-based Regression Test Case Prioritization. *SIGSOFT Softw. Eng. Notes*, 35(6), 1-7.
<https://doi.org/10.1145/1874391.1874405>
- Panigrahi, C. R. & Mall, R. (2012). A hybrid regression test selection technique for object-oriented programs. *Int. J. Soft. Eng. Applic*, 6(4), 17-34.
- Parsa, S. & Khalilian, A. (2009). A Bi-objective Model Inspired Greedy Algorithm for Test Suite Minimization. *In Future Generation Information Technology*, 208-215.
- Parsa, S., & Khalilian, A. (2009). A Bi-objective Model Inspired Greedy Algorithm for Test Suite Minimization. En Y. Lee, T. Kim, W. Fang, & D. Ślęzak (Eds.), *Future Generation Information Technology* (pp. 208-215). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-10509-8_24
- Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, 68-77.
<http://dl.acm.org/citation.cfm?id=2227115.2227123>
- Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., & Wang, B. (2003). Automated support for classifying software failure reports. *25th International Conference on Software Engineering, 2003. Proceedings.*, 465-475. <https://doi.org/10.1109/ICSE.2003.1201224>
- Podgurski, Andy, & Yang, C. (1993). Partition Testing, Stratified Sampling, and Cluster Analysis. *Proceedings of the 1st ACM SIGSOFT*

Symposium on Foundations of Software Engineering, 169-181.

<https://doi.org/10.1145/256428.167076>

Pravin. (2013). EFFECTIVE TEST CASE SELECTION AND PRIORITIZATION IN REGRESSION TESTING. *Journal of Computer Science*, 9(5), 654-659. <https://doi.org/10.3844/jcssp.2013.654.659>

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education UK.

Rafaqut Kazmi, Dayang N. A. Jawawi, Radziah Mohamad, and Imran Ghani.

(2017). Effective Regression Test Case Selection: A Systematic Literature Review. *ACM Comput. Surv.* 50, 2, Article 29.

[https://doi.org/DOI: https://doi.org/10.1145/3057269](https://doi.org/DOI:https://doi.org/10.1145/3057269)

Rogers, R. O. (2004). Scaling Continuous Integration. En J. Eckstein & H. Baumeister (Eds.), *Extreme Programming and Agile Processes in Software Engineering* (pp. 68-76). Springer Berlin Heidelberg.

https://doi.org/10.1007/978-3-540-24853-8_8

Rogstad, E., & Briand, L. C. (2015). Clustering Deviations for Black Box Regression Testing of Database Applications. *IEEE Transactions on Reliability*, PP(99), 1-15. <https://doi.org/10.1109/TR.2015.2437840>

Rogstad, E., & Briand, L. C. (2016). Clustering Deviations for Black Box Regression Testing of Database Applications. *IEEE Transactions on Reliability*, 65(1), 4-18. <https://doi.org/10.1109/TR.2015.2437840>

Rogstad, E., Briand, L., Dalberg, R., Rynning, M., & Arisholm, E. (2011). Industrial experiences with automated regression testing of a legacy database application. *2011 27th IEEE International Conference on*

Software Maintenance (ICSM), 362-371.

<https://doi.org/10.1109/ICSM.2011.6080803>

Rogstad, Erik, & Briand, L. (2016). Cost-effective strategies for the regression testing of database applications: Case study and lessons learned. *Journal of Systems and Software*, 113, 257-274.

<https://doi.org/10.1016/j.jss.2015.12.003>

Rogstad, Erik, Briand, L., & Torkar, R. (2013). Test case selection for black-box regression testing of database applications. *Information and Software Technology*, 55(10), 1781-1795.

<https://doi.org/10.1016/j.infsof.2013.04.004>

Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016). 15 Years of Software Regression Testing Techniques—A Survey. *International Journal of Software Engineering and Knowledge Engineering*, 26(05), 675-689.

<https://doi.org/10.1142/S0218194016300013>

Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016c). 15 Years of Software Regression Testing Techniques—A Survey. *International Journal of Software Engineering and Knowledge Engineering*, 26(05), 675-689.

<https://doi.org/10.1142/S0218194016300013>

Rothermel, G., & Harrold, M. J. (1993). A safe, efficient algorithm for regression test selection. , *Conference on Software Maintenance ,1993. CSM-93, Proceedings*, 358-367.

<https://doi.org/10.1109/ICSM.1993.366926>

Rothermel, G., Harrold, M. J., Ostrin, J., & Hong, C. (1998). An empirical study of the effects of minimization on the fault detection capabilities of

- test suites. , *International Conference on Software Maintenance, 1998. Proceedings*, 34-43. <https://doi.org/10.1109/ICSM.1998.738487>
- Rothermel, Gregg, & Harrold, M. J. (1997). A Safe, Efficient Regression Test Selection Technique. *ACM Trans. Softw. Eng. Methodol.*, 6(2), 173-210. <https://doi.org/10.1145/248233.248262>
- Runeson, P., & Zelkowitz, M. (2005). Integrating agile software development into stage-gate managed product development. *Empirical Software Engineering*, 11.
- Saha, G. K. (2008). Understanding Software Testing Concepts. *Ubiquity*, 2008(February), 2:1-2:1. <https://doi.org/10.1145/1348483.1348484>
- Sangwan, R. S., & LaPlante, P. A. (2006). Test-Driven Development in Large Projects. *IT Professional*, 8(5), 25-29. <https://doi.org/10.1109/MITP.2006.122>
- Shi, A., Thummalapenta, S., Lahiri, S. K., Bjorner, N., & Czerwonka, J. (2017). Optimizing Test Placement for Module-level Regression Testing. *Proceedings of the 39th International Conference on Software Engineering*, 689-699. <https://doi.org/10.1109/ICSE.2017.69>
- Soetens, Q. D., Demeyer, S., & Zaidman, A. (2013). Change-Based Test Selection in the Presence of Developer Tests. *2013 17th European Conference on Software Maintenance and Reengineering (CSMR)*, 101-110. <https://doi.org/10.1109/CSMR.2013.20>
- Steinert, B., Haupt, M., Krahn, R., & Hirschfeld, R. (2010). Continuous Selective Testing. In A. Sillitti, A. Martin, X. Wang, & E. Whitworth (Eds.), *Agile Processes in Software Engineering and Extreme*

Programming (pp. 132-146). Springer Berlin Heidelberg.

http://link.springer.com/chapter/10.1007/978-3-642-13054-0_10

Suarez-Cabal, M. J., & Tuya, J. (2005). Coverage Measurement for SQL

Queries. *IEEE Latin America Transactions*, 3(1), 49-55.

<https://doi.org/10.1109/TLA.2005.1468662>

Tuya, J., Dolado, J., Suarez-Cabal, M. J., & de la Riva, C. (2008). A

Controlled Experiment on White-box Database Testing. *SIGSOFT*

Softw. Eng. Notes, 33(1), 8:1-8:6.

<https://doi.org/10.1145/1344452.1344462>

Vangala, V., Czerwonka, J., & Talluri, P. (2009). Test Case Comparison and

Clustering Using Program Profiles and Static Execution. *Proceedings*

of the the 7th Joint Meeting of the European Software Engineering

Conference and the ACM SIGSOFT Symposium on The Foundations

of Software Engineering, 293-294.

<https://doi.org/10.1145/1595696.1595748>

Vicente, E., Anticon, M. T., & Rivera, J. (2007). *GraspKM en la Recuperación*

de la Estructura de Software. 35-42.

Vidgen, R., & Wang, X. (2009). Coevolving Systems and the Organization of

Agile Software Development. *Information Systems Research*, 20(3),

355-376. <https://doi.org/10.1287/isre.1090.0237>

Vidyanand, C., & Ramesh, G. (2017). Improving the Performance of

Database Applications by using Black Box Regression Testing.

CSEIT172533 | Received : 08 Sep 2017 | Accepted : 14 Sep 2017 |

September - October - 2017 [(2) 4 : 158 - 162] International Journal

of Scientific Research in Computer Science, Engineering and Information Technology, Volume 2(5).

Wagner, S., & Seifert, T. (2005). Software Quality Economics for Defect-detection Techniques Using Failure Prediction. *Proceedings of the Third Workshop on Software Quality*, 1-6.

<https://doi.org/10.1145/1083292.1083296>

Wallace, D. R., & Fujii, R. U. (1989). Software verification and validation: An overview. *IEEE Software*, 6(3), 10-17.

<https://doi.org/10.1109/52.28119>

Webster, J., & Watson, R. T. (2002). Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly*, 26(2), xiii-xxiii.

Wikstrand, G., Feldt, R., Gorantla, J. K., Zhe, W., & White, C. (2009a).

Dynamic Regression Test Selection Based on a File Cache An Industrial Evaluation. *2009 International Conference on Software Testing Verification and Validation*, 299-302.

<https://doi.org/10.1109/ICST.2009.42>

Wikstrand, G., Feldt, R., Gorantla, J. K., Zhe, W., & White, C. (2009b).

Dynamic Regression Test Selection Based on a File Cache An Industrial Evaluation. *International Conference on Software Testing Verification and Validation, 2009. ICST '09*, 299-302.

<https://doi.org/10.1109/ICST.2009.42>

Willmor, D., & Embury, S. M. (2005). A safe regression test selection technique for database-driven applications. *21st IEEE International Conference on Software Maintenance (ICSM'05)*, 421-430.

<https://doi.org/10.1109/ICSM.2005.15>

- Witten, I. H., & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012a). *Experimentation in Software Engineering*. Springer Science & Business Media.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012b). *Experimentation in Software Engineering*. Springer Science & Business Media.
- Yan, S., Chen, Z., Zhao, Z., Zhang, C., & Zhou, Y. (2010). A Dynamic Test Cluster Sampling Strategy by Leveraging Execution Spectra Information. *2010 Third International Conference on Software Testing, Verification and Validation (ICST)*, 147-154.
<https://doi.org/10.1109/ICST.2010.47>
- Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, 22(2), 67-120. <https://doi.org/10.1002/stvr.430>
- Yoo, Shin, & Harman, M. (2007). Pareto Efficient Multi-objective Test Case Selection. *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, 140-150.
<https://doi.org/10.1145/1273463.1273483>
- Yoo, Shin, Harman, M., Tonella, P., & Susi, A. (2009). Clustering Test Cases to Achieve Effective and Scalable Prioritisation Incorporating Expert Knowledge. *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, 201-212.
<https://doi.org/10.1145/1572272.1572296>

- Zhang, C., Chen, Z., Zhao, Z., Yan, S., Zhang, J., & Xu, B. (2010). An Improved Regression Test Selection Technique by Clustering Execution Profiles. *2010 10th International Conference on Quality Software (QSIC)*, 171-179. <https://doi.org/10.1109/QSIC.2010.16>
- Zhang, D., Zhou, Z.-H., & Chen, S. (2007). Semi-Supervised Dimensionality Reduction. En C. Apte, D. Skillicorn, B. Liu, & S. Parthasarathy (Eds.), *Proceedings of the 2007 SIAM International Conference on Data Mining* (pp. 629-634). Society for Industrial and Applied Mathematics. <http://epubs.siam.org/doi/abs/10.1137/1.9781611972771.73>
- Zhang, J., Gao, J., Zhou, M., & Wang, J. (2017). Improving the Effectiveness of Information Retrieval with Clustering and Fusion. *Computational Linguistics and Chinese Language Processing*, 6/1. <https://www.microsoft.com/en-us/research/publication/improving-effectiveness-information-retrieval-clustering-fusion/>
- Zhou, Z. Q., Sinaga, A., Susilo, W., Zhao, L., & Cai, K.-Y. (2018). A cost-effective software testing strategy employing online feedback information. *Information Sciences*, 422, 318-335. <https://doi.org/10.1016/j.ins.2017.08.088>

ANEXO A: PUBLICACIONES RESULTADOS DE LA TESIS

A continuación, se detalla las publicaciones en revistas de impacto (Q1 y Q3), (WOS Web of Science y SCOPUS) realizadas como parte del desarrollo del trabajo:

- Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016). 15 Years of Software Regression Testing Techniques — A Survey. *International Journal of Software Engineering and Knowledge Engineering*, 26(05), 675–689.
- Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2017). An Approach for Regression Testing of Database Applications in Incremental Development Settings. In *2017 6th International Conference on Software Process Improvement (CIMPS)* (pp. 1–4).
- Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2017). Regression Testing of Database Applications Under an Incremental Software Development Setting. *IEEE Access*, 5, 18419–18428.

ANEXO B: PLAN DE PRUEBAS: SOFTWARE ACADÉMICO**PLAN DE PRUEBAS UNITARIAS CON ACCESO A DATOS****SISTEMA ACADÉMICO****Historia de Revisiones**

Fecha	Versión	Descripción	Autor
02-09-2018	1.0	Creación del documento	ADMIN RRM
04-09-2018	1.0	Creación Plan de Prueba en Testlink	ADMIN RRM
08-09-2018	1.0	Ejecución del Plan de prueba de Académico versión 1 por Testlink	ADMIN RRM
10-09-2018	1.0	Registro de ejecución de casos de prueba registrado por Testlink	ADMIN RRM
10-09-2018	1.0	Generación de Plan de Pruebas	ADMIN RRM
14-09-2018	1.0	Documentación de casos de prueba	ADMIN RRM

Contenido

HISTORIA DE REVISIONES.....	170
INTRODUCCIÓN.....	172
OBJETIVO	172
ESTRATEGIA DE LAS PRUEBAS	172
ALCANCE	173
ARTIFACTOS DE PRUEBA	173
CARACTERÍSTICAS A SER PROBADAS	173
CRITERIOS DE ENTRADA Y SALIDA	174
ESPECIFICACIÓN DE REQUISITOS	174
EJECUCIÓN DEL PLAN DE PRUEBAS.....	177
SUITE DE PRUEBAS: GESTIÓN DE AJUSTES	177

Introducción

A continuación, se muestra de manera representativa el Plan de Pruebas Unitarias con acceso a datos, de la primera versión del módulo 1: Gestión de catálogos - Gestión de ajustes del producto software “Sistemas Académico”, a fin de que se pueda realizar la verificación y validación del producto, el Plan completo se encuentra en la siguiente dirección: <http://raulrosero.ml>, con las credenciales user=admin y password=admin. Por otro lado, también se presenta el resultado por versiones de desarrollo, este proceso ha sido llevado de manera automatizada por la herramienta para gestión de planes de prueba de software open source: TestLink.

Objetivo

El propósito del plan de pruebas es proveer la información para gestionar, planear, ejecutar y controlar las ejecuciones del plan de pruebas a través de sus casos de prueba unitarias con acceso a datos de un proyecto.

Estrategia de las Pruebas

Para ello se ha utilizado como base los documentos de los proyectos: Estafeta, Académico y Académico y los casos de prueba con acceso a datos que serán considerados en las pruebas de regresión, de tal manera que aseguren la calidad de los productos.

Alcance

Los planes de prueba de los productos se realizarán de manera incremental, por versión como se desarrollaron los productos software.

Artefactos de prueba

Los módulos o componentes que fueron considerados en cada versión son los siguientes:

Versión 1: Gestión de Catálogos

1.1: Gestión de Ajustes

1.1.1: Fajustes.intInsertar

1.1.2: Fajustes obtener

1.1.3: Fajustes actualizar

1.1.4: Fajustes eliminar

1.1.5: Fajustes obtenerdadocodigo

Características para comprobar

Las características ser probadas son de carácter funcional de acuerdo con los casos de éxito fracaso del caso de prueba con acceso a datos, lo que se describe en la documentación de casos de prueba gestionado por TestLink.

Criterios de entrada y salida

- Conjunto de casos de prueba con acceso a datos.
- Claridad en el procedimiento para la ejecución de los casos de prueba unitarias de acuerdo con la documentación gestionada por TestLink.
- El entorno de pruebas debe ser el adecuado para la realización de las pruebas, en este caso fue en el mismo ambiente de desarrollo.

Entrada

- Todas las pruebas se deben ejecutar sin errores para liberar la versión.

Salida

- Un componente principal tiene un error que impide probar un área importante.
- El entorno de pruebas no es lo suficientemente estable como para confiar en los resultados.
- El entorno de pruebas es muy diferente del entorno de producción previsto y no se puede confiar en los resultados.

Especificación de requisitos de Académico: iteración 1

Iteración # 1 con 45 casos de prueba con acceso a datos de la versión 1 del sistema de seguimiento de Académico docente.1.1. Suite de Pruebas: Casos de prueba con acceso a datos de la Versión 1 de Académico.

1. Espec. de Requisitos: Versión 1: Gestión de Catálogos	
Revisión	3
Autor	admin
Tipo	Especificación de Requisito de Usuario
Sobrecribir cuenta Requisitos (Cobertura)	la de N/A (0/0)

1.1. Espec. de Requisitos: 1: Gestión de Ajustes	
Revisión	2
Autor	admin
Tipo	Especificación de Requisito de Usuario
Sobrecribir cuenta Requisitos (Cobertura)	la de N/A (5/0)

Requisito: 1.1.1: FAjustes.intlInsertar	
Versión	1
Revisión	7
Autor	admin
Estado	Válido
Tipo	Caso de Uso
Cobertura	100% (1/1)
Casos de Prueba Relacionados	2 : FAjustes.intlInsertar [1]

Requisito: 1.1.2: Fajustes obtener	
Versión	1
Revisión	4
Autor	admin
Estado	Válido

Tipo	Caso de Uso
Cobertura	100% (1/1)
Casos de Prueba Relacionados	3 : Fajustes obtener [1]

Requisito: 1.1.3: Fajustes actualizar	
Versión	1
Revisión	4
Autor:	admin
Estado	Válido
Tipo	Caso de Uso
Cobertura	100% (1/1)
Casos de Prueba Relacionados	4 : Fajustes actualizar [1]

Requisito: 1.1.4: Fajustes eliminar	
Versión	1
Revisión	4
Autor	admin
Estado	Válido
Tipo	Caso de Uso
Cobertura	100% (1/1)
Casos de Prueba Relacionados	5 : Fajustes eliminar [1]

Requisito: 1.1.5: Fajustes obtenerdadocodigo	
Versión	1
Revisión	4
Autor	admin
Estado	Válido
Tipo	Caso de Uso
Cobertura	100% (1/1)
Casos de Prueba Relacionados	6: Fajustes obtenerdadocodigo [1]

Ejecución del plan de pruebas

Casos de prueba unitarios con acceso a datos del producto software “sistema Académico” en su 1.^a Versión/iteración.

Suite de Pruebas: Gestión de Ajustes

Este ambiente permite la gestión de los ajustes relacionados a la funcionalidad operativa del sistema.

CPA1: Fajustes.intInsertar

CPA1.2: Fajustes obtener

CPA1.3: Fajustes actualizar

CPA1.4: Fajustes eliminar

CPA1.5: Fajustes obtenerdadocodigo

Tabla 1. Caso de prueba: FAjustes.intInsertar

Caso de Prueba CPA-2: FAjustes.intInsertar [1]				
Autor:		admin		
Resumen:				
Requerimiento: "FAjustes.intInsertar" (version 1).				
<pre>@Test public void testintInsertar() throws Exception { System.out.println("intInsertar"); String metodo = "intInsertar"; String clase = "FAjustes"; Date inicio = new Date(); Ajustes param1 = new Ajustes(); param1.setDescripcion("NP"); param1.setPeriodo(new Periodo(2)); param1.setValor(4); Boolean expResult = true; Integer result = 0; try { result = FAjustes.intInsertar(param1); } catch (Exception e) { result = 0; } Boolean res = false; if (result != 0) { res = true; } else { res = false; } //assertEquals(expResult, res); Date vfinal = new Date(); long duracion = Util.DiferenciaFechas(vinicio, vfinal); Util.exportar(clase, metodo, duracion);assertEquals(expResult, res); }</pre>				
Precondiciones: Debe existir una conexión funcional hacia la Base de Datos				
N.º:	Pasos:	Resultados Esperados:	Notas de la ejecución:	Estado de la ejecución:
1	Registrar los parámetros correctos requeridos por el método a Probar (Objeto a Insertar)	Valor entero que representa el Id Asignado al registro Insertado		Pasado
Tipo de ejecución:	Automatizada			
Duración estimada de la ejec. (min):				
Prioridad:	Media			
Responsable:	Raúl Rosero			
Fecha:				
Requisitos:	1.1: FAjustes.intInsertar			

Keywords:	Ninguno
Detalles de la ejecución:	
Build:	3. Plan de Pruebas Académico versión 1
Tester:	admin
Resultado de la Ejecución:	Pasado
Modo de Ejecución:	Manual
Duración de la ejecución (min):	1.00

Tabla 2 Caso de prueba: FAjustes.obtener

Caso de Prueba CPA-3: Fajustes obtener [1]				
Autor:		admin		
Resumen:				
Requerimiento: "Fajustes obtener" (version 1).				
<pre>@Test public void testobtener() throws Exception { System.out.println("obtener"); String metodo = "obtener"; String clase = "FAjustes"; Date inicio = new Date(); Boolean expResult = true; ArrayList<Ajustes> result = null; try { result = FAjustes.obtener(); } catch (Exception e) { result = null; } Boolean res = false; if (result != null) { res = true; } else { res = false; } //assertEquals(expResult, res); Date vfinal = new Date(); long duracion = Util.DiferenciaFechas(inicio, vfinal); Util.exportar(clase, metodo, duracion);assertEquals(expResult, res); }</pre>				
Precondiciones: Debe existir una conexión funcional hacia la Base de Datos.				
N.º:	Pasos:	Resultados Esperados:	Notas de la ejecución:	Estado de la ejecución:
1	Obtener Definir el tipo de dato a recibir desde la base (List<? Extends Object>)	Colección de Objetos existentes en la consulta		

		(List<? extends Object> != null)		
<u>Tipo de ejecución:</u>	Automatizada			
<u>Duración estimada de la ejec. (min):</u>				
<u>Prioridad:</u>	Media			
<u>Responsable:</u>	Raúl Rosero			
<u>Fecha:</u>				
<u>Requisitos:</u>	1.2: Fajustes obtener			
<u>Keywords:</u>	Ninguno			
Detalles de la ejecución:				
<u>Build:</u>	3. Plan de Pruebas Académico versión 1			
<u>Tester:</u>	admin			
<u>Resultado de la Ejecución:</u>	Pasado			
<u>Modo de Ejecución:</u>	Manual			
<u>Duración de la ejecución (min):</u>	1.00			

Tabla 3 Caso de prueba: FAjustes.actualizar

Caso de Prueba CPA-4: Fajustes actualizar [1]	
<u>Autor:</u>	admin
<u>Resumen:</u> Requerimiento: "Fajustes actualizar" (version 1). <pre>@Test public void testActualizar() throws Exception { System.out.println("Actualizar"); String metodo = "Actualizar"; String clase = "FAjustes"; Date inicio = new Date(); Ajustes param1 = new Ajustes(); param1.setCodigo(3); param1.setDescripcion("NP"); param1.setPeriodo(new Periodo(2)); param1.setValor(4); Boolean expResult = true; Boolean result = false; try {</pre>	

<pre> result = FAjustes.Actualizar(param1); } catch (Exception e) { result = false; } Boolean res = false; if (result != false) { res = true; } else { res = false; } //assertEquals(expResult, res); Date vfinal = new Date(); long duracion = Util.DiferenciaFechas(vinicio, vfinal); Util.exportar(clase, metodo, duracion);assertEquals(expResult, res); } </pre>				
Precondiciones: Debe existir una conexión funcional hacia la Base de Datos.				
N.º:	Pasos:	Resultados Esperados:	Notas de la ejecución:	Estado de la ejecución:
1	Registrar los parámetros correctos requeridos por el método a Probar (Objeto a Actualizar)	Valor Boolean que representa el éxito de la actualización		
Tipo de ejecución:	Automatizada			
Duración estimada de la ejec. (min):				
Prioridad:	Media			
Responsable:	Raúl Rosero			
Fecha:				
Requisitos:	1.3: Fajustes actualizar			
Keywords:	Ninguno			
Detalles de la ejecución:				
Build:	3. Plan de Pruebas Académico versión 1			
Tester:	admin			
Resultado de la Ejecución:	Pasado			
Modo de Ejecución:	Manual			
Duración de la ejecución (min):	1.00			

Tabla 4 Caso de prueba: FAjustes.eliminar

Caso de Prueba CPA-5: Fajustes eliminar [1]				
<u>Autor:</u>	admin			
<u>Resumen:</u>	Requerimiento: "Fajustes eliminar" (version 1). <pre> @Test public void testEliminar() throws Exception { System.out.println("Eliminar"); String metodo = "Eliminar"; String clase = "FAjustes"; Date inicio = new Date(); Ajustes param1 = new Ajustes(); param1.setCodigo(16); Boolean expectedResult = true; Boolean result = false; try { result = FAjustes.Eliminar(param1); } catch (Exception e) { result = false; } Boolean res = false; if (result != false) { res = true; } else { res = false; } //assertEquals(expResult, res); Date vfinal = new Date(); long duracion = Util.DiferenciaFechas(inicio, vfinal); Util.exportar(clase, metodo, duracion);assertEquals(expResult, res); } </pre>			
<u>Precondiciones:</u>	Debe existir una conexión funcional hacia la Base de Datos.			
<u>N.º:</u>	<u>Pasos:</u>	<u>Resultados Esperados:</u>	<u>Notas de la ejecución:</u>	<u>Estado de la ejecución:</u>
1	Registrar los parámetros correctos requeridos por el método a Probar (Objeto a Eliminar)	Valor Boolean que representa el éxito de la eliminación		
<u>Tipo de ejecución:</u>	Automatizada			
<u>Duración estimada de la ejec. (min):</u>				
<u>Prioridad:</u>	Media			
<u>Responsable:</u>	Raúl Rosero			
<u>Fecha:</u>				
<u>Requisitos:</u>	1.4: Fajustes eliminar			
<u>Keywords:</u>	Ninguno			

Detalles de la ejecución:	
Build:	3. Plan de Pruebas Académico versión 1
Tester:	admin
Resultado de la Ejecución:	Pasado
Modo de Ejecución:	Manual
Duración de la ejecución (min):	1.00

Tabla 5 Caso de prueba: Fajustes.obtenerdadocodigo

Caso de Prueba CPA-6: Fajustes obtenerdadocodigo [1]				
Autor:		admin		
Resumen:				
Requerimiento: "Fajustes obtenerdadocodigo" (version 1). <pre>@Test public void testobtenerdadocodigo() throws Exception { System.out.println("obtenerdadocodigo"); String metodo = "obtenerdadocodigo"; String clase = "FAjustes"; Date inicio = new Date(); int param1 = 3; Boolean expResult = true; Ajustes result = null; try { result = FAjustes.obtenerdadocodigo(param1); } catch (Exception e) { result = null; } Boolean res = false; if (result != null) { res = true; } else { res = false; } //assertEquals(expResult, res); Date vfinal = new Date(); long duracion = Util.DiferenciaFechas(inicio, vfinal); Util.exportar(clase, metodo, duracion);assertEquals(expResult, res); }</pre>				
Precondiciones: Debe existir una conexión funcional hacia la Base de Datos.				
N.º:	Pasos:	Resultados Esperados:	Notas de la ejecución:	Estado de la ejecución:
1	Definir el tipo de dato a recibir desde la base (List<? Extends Object>)	Objeto existente en la consulta (? extends Object != null)		

<u>Tipo de ejecución:</u>	Automatizada
<u>Duración estimada de la ejec. (min):</u>	
<u>Prioridad:</u>	Media
<u>Responsable:</u>	Raúl Rosero
<u>Fecha:</u>	
<u>Requisitos:</u>	1.5: Fajustes obtenerdadocodigo
<u>Keywords:</u>	Ninguno
Detalles de la ejecución:	
<u>Build:</u>	3. Plan de Pruebas Académico versión 1
<u>Tester:</u>	admin
<u>Resultado de la Ejecución:</u>	Pasado
<u>Modo de Ejecución:</u>	Manual
<u>Duración de la ejecución (min):</u>	1.00

ANEXO C: ANALISIS DEL IMPACTO DE FALLOS EN LAS PRUEBAS DE REGRESIÓN

El análisis se lo realiza con la siguiente nomenclatura:

Tabla 1 Terminología para tabulación de clusterización en cada producto software

seed	Semillas para proceso de clusterización
K	Número de clústeres
T	Conjunto total de casos de prueba
T _n	Conjunto de casos de prueba de la versión
T'	Subconjunto de casos de prueba seleccionados en la versión
T _f	Conjunto de casos de prueba fallos
T' _f	Subconjunto de casos de prueba fallos
TR	capacidad de reducción de casos de prueba en cada versión
R	Recuperación o capacidad de detección de fallos
P	Precisión
R	Rendimiento combinado

Producto Software Estafeta

Versión 2:

Los casos de prueba fallidos para esta versión fueron 8, los mismos que son T_f = (7, 8, 10, 11, 32, 58, 70, 72) del conjunto de 72 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 3, a los mismos que agruparon en un promedio de 62 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 2 Registro de datos de la prueba de regresión de la versión 2 de Estafeta

seed	K	T	T _n	T'	T _f	T' _f	TR	R	P	RC
1	3	72	43	69	8	8	4.2%	100%	12%	0.21
10	5	72	43	61	8	8	15.3%	100%	13%	0.23
50	4	72	43	56	8	8	22.2%	100%	14%	0.25
100	3	72	43	56	8	8	22.2%	100%	14%	0.25
150	2	72	43	72	8	8	0.0%	100%	11%	0.20
200	5	72	43	67	8	8	6.9%	100%	12%	0.21
250	6	72	43	49	8	8	31.9%	100%	16%	0.28
500	4	72	43	72	8	8	0.0%	100%	11%	0.20
750	3	72	43	61	8	8	15.3%	100%	13%	0.23
1000	4	72	43	49	8	8	31.9%	100%	16%	0.28

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla3. Comparativo de las técnicas en la versión 2 de Estafeta

Cuadro comparativo de resultados de aplicación de los métodos en la versión 2 de Estafeta		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	15%	0%
Recuperación (Recall)	100%	100%
Precision	13%	10%
Rendimiento combinado	0.23	0,11

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.12, con una cantidad de reducción de casos de prueba de 15%.

Versión 3:

Los casos de prueba fallidos para esta versión fueron 11, los mismos que son $T_f = (5, 9, 10, 14, 15, 70, 73, 80, 89, 121, 131)$ del conjunto de 134 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 4, los mismos que agruparon en un promedio de 126 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 4. Registro de la prueba de regresión de la versión 3 de Estafeta

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	3	134	62	121	11	11	9.7%	100%	9%	0.17
10	3	134	62	132	11	11	1.5%	100%	8%	0.15
50	3	134	62	132	11	11	1.5%	100%	8%	0.15
100	2	134	62	121	11	11	9.7%	100%	9%	0.17
150	3	134	62	132	11	11	1.5%	100%	8%	0.15
200	5	134	62	121	11	11	9.7%	100%	9%	0.17
250	3	134	62	132	11	11	1.5%	100%	8%	0.15
500	4	134	62	116	11	11	13.4%	100%	9%	0.17
750	2	134	62	121	11	11	9.7%	100%	9%	0.17
1000	3	134	62	132	11	11	1.5%	100%	8%	0.15

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 5. Comparativo de las técnicas en la versión 3 de Estafeta

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	6%	0%
Recuperación (Recall)	100%	100%
Precision	9%	9%
Rendimiento combinado	0.16	0,14

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.02, con una capacidad de reducción del 6%.

Versión 4:

Los casos de prueba fallidos para esta versión fueron 9, los mismos que son $T_f = (29, 47, 51, 60, 94, 98, 114, 119, 143)$ del conjunto de 163 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 6, los mismos que agruparon en un promedio de 132 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 6. Registro de la prueba de regresión de la versión 4 de Estafeta

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	3	163	29	148	9	9	9.2%	100%	6%	0.11
10	3	163	29	147	9	9	9.8%	100%	6%	0.12
50	3	163	29	122	9	9	25.2%	100%	7%	0.14
100	4	163	29	33	9	9	79.8%	100%	27%	0.43
150	4	163	29	154	9	9	5.5%	100%	6%	0.11
200	7	163	29	121	9	9	25.8%	100%	7%	0.14
250	6	163	29	134	9	9	17.8%	100%	7%	0.13
500	5	163	29	154	9	9	5.5%	100%	6%	0.11
750	6	163	29	147	9	9	9.8%	100%	6%	0.12
1000	7	163	29	148	9	9	9.2%	100%	6%	0.11

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 7. Comparativo de las técnicas en la versión 4 de Estafeta

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	20%	0%
Recuperación (Recall)	100%	100%
Precision	8%	6%
Rendimiento combinado	0.15	0,11

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.04, con una capacidad de reducción del 20%.

Versión 5:

Los casos de prueba fallidos para esta versión fueron 10, los mismos que son Tf = (52, 57, 66, 69, 106, 114, 120, 181, 184, 196) del conjunto de 199 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 4, los mismos que agruparon en un promedio de 175 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 8. Registro de la prueba de regresión de la versión 5 de Estafeta

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	2	199	36	190	10	10	4.5%	100%	5%	0.10
10	4	199	36	195	10	10	2.0%	100%	5%	0.10
50	4	199	36	187	10	10	6.0%	100%	5%	0.10
100	4	199	36	35	10	10	82.4%	100%	29%	0.44
150	5	199	36	195	10	10	2.0%	100%	5%	0.10
200	6	199	36	190	10	10	4.5%	100%	5%	0.10
250	7	199	36	114	10	10	42.7%	100%	9%	0.16
500	2	199	36	190	10	10	4.5%	100%	5%	0.10
750	5	199	36	197	10	10	1.0%	100%	5%	0.10
1000	4	199	36	186	10	10	6.5%	100%	5%	0.10

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 9. Comparativo de las técnicas en la versión 5 de Estafeta

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	16%	0%
Recuperación (Recall)	100%	100%
Precision	8%	5%
Rendimiento combinado	0.14	0,09

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.05, con una capacidad de reducción del 16%.

a. Impacto Global de los fallos en desarrollo del producto Estafeta

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante las técnicas consideradas, en las diferentes versiones de Estafeta, se obtiene el siguiente cuadro:

Tabla10. Comparativo global de las técnicas en el producto software Estafeta

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	14%	0%
Recuperación (Recall)	100%	100%
Precision	10%	7%
Rendimiento combinado	0.17	0,11

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización, con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima al valor recomendado de 1 del rendimiento combinado, considerando la cantidad de reducción de casos de prueba del 14%

Producto Software Silabo

Versión 2:

Los casos de prueba fallidos para esta versión fueron 8, los mismos que son $T_f = (8, 10, 12, 14, 18, 19, 41, 42)$ del conjunto de 60 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 42 (T') casos de prueba para la prueba de regresión de la versión.

Tabla11. Registro de la prueba de regresión de la versión 2 de Silabo

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	5	60	33	44	8	8	26.7%	100%	18%	0.31
10	6	60	33	49	8	8	18.3%	100%	16%	0.28
50	6	60	33	37	8	8	38.3%	100%	22%	0.36
100	3	60	33	21	8	8	65.0%	100%	38%	0.55
150	5	60	33	33	8	8	45.0%	100%	24%	0.39
200	6	60	33	49	8	8	18.3%	100%	16%	0.28
250	5	60	33	43	8	8	28.3%	100%	19%	0.31
500	4	60	33	54	8	8	10.0%	100%	15%	0.26
750	5	60	33	49	8	8	18.3%	100%	16%	0.28
1000	8	60	33	43	8	8	28.3%	100%	19%	0.31

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla12. Comparativo de las técnicas en la versión 2 de Silabo

Cuadro comparativo de resultados de aplicación de los métodos en la versión 2 de Silabo		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	30%	0%
Recuperación (Recall)	100%	100%
Precision	21%	13%
Rendimiento combinado	0.34	0,23

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.11, con una capacidad de reducción del 30%.

Versión 3:

Los casos de prueba fallidos para esta versión fueron 10, los mismos que son $T_f = (17, 19, 23, 44, 61, 41, 84, 65, 62, 63,)$ del conjunto de 89 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 65 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 13. Registro de la prueba de regresión de la versión 3 de Silabo

seed	K	T	Tn	T'	T _f	T' _f	TR	R	P	RC
1	3	89	29	73	10	10	18.0%	100%	14%	0.24
10	5	89	29	83	10	10	6.7%	100%	12%	0.22
50	7	89	29	50	10	10	43.8%	100%	20%	0.33
100	4	89	29	26	10	10	70.8%	100%	38%	0.56
150	6	89	29	71	10	10	20.2%	100%	14%	0.25
200	5	89	29	89	10	10	0.0%	100%	11%	0.20
250	3	89	29	81	10	10	9.0%	100%	12%	0.22
500	3	89	29	72	10	10	19.1%	100%	14%	0.24
750	4	89	29	79	10	10	11.2%	100%	13%	0.22
1000	5	89	29	34	10	10	61.8%	100%	29%	0.45

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 14. Comparativo de las técnicas en la versión 3 de Silabo

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	26%	0%
Recuperación (Recall)	100%	100%
Precision	18%	11%
Rendimiento combinado	0.29	0,22

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.17 con una capacidad de reducción de casos de prueba del 26%.

Versión 4:

Los casos de prueba fallidos para esta versión fueron 6, los mismos que son $T_f = (64, 71, 77, 84, 94, 109,)$ del conjunto de 119 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 45 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 15. Registro de la prueba de regresión de la versión 4 de Silabo

seed	K	T	T_n	T'	T_f	T'_f	TR	R	P	RC
1	2	119	30	25	6	6	100.0%	100%	24%	0.39
10	5	119	30	25	6	6	79.0%	100%	24%	0.39
50	6	119	30	25	6	6	79.0%	100%	24%	0.39
100	6	119	30	22	6	6	81.5%	100%	27%	0.43
150	5	119	30	32	6	6	73.1%	100%	19%	0.32
200	4	119	30	61	6	6	48.7%	100%	10%	0.18
250	3	119	30	99	6	6	16.8%	100%	6%	0.11
500	3	119	30	108	6	6	9.2%	100%	6%	0.11
750	3	119	30	31	6	6	73.9%	100%	19%	0.32
1000	7	119	30	25	6	6	79.0%	100%	24%	0.39

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 16. Comparativo de las técnicas en la versión 4 de Silabo

Cuadro comparativo de resultados de aplicación de los métodos en la versión 4 de Silabo		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	64%	0%
Recuperación (Recall)	100%	100%
Precision	18%	5%
Rendimiento combinado	0.29	0,09

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.20, con una capacidad de reducción de casos de prueba del 64%.

Versión 5:

Los casos de prueba fallidos para esta versión fueron 5, los mismos que son $T_f = (63, 129, 122, 124, 155)$ del conjunto de 158 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 79 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 17. Registro de la prueba de regresión de la versión 5 de Silabo

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	4	158	39	133	5	5	15.8%	100%	4%	0.07
10	6	158	39	32	5	5	79.7%	100%	16%	0.27
50	4	158	39	39	5	5	75.3%	100%	13%	0.23
100	5	158	39	35	5	5	77.8%	100%	14%	0.25
150	5	158	39	134	5	5	15.2%	100%	4%	0.07
200	6	158	39	39	5	5	75.3%	100%	13%	0.23
250	5	158	39	85	5	5	46.2%	100%	6%	0.11
500	5	158	39	115	5	5	27.2%	100%	4%	0.08
750	3	158	39	142	5	5	10.1%	100%	4%	0.07
1000	4	158	39	44	5	5	72.2%	100%	11%	0.20

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 18. Comparativo de las técnicas en la versión 5 de Silabo

Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	50%	0%
Recuperación (Recall)	100%	100%
Precision	9%	5%
Rendimiento combinado	0.16	0,05

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.11, con una capacidad de reducción de casos de prueba del 50%.

Versión 6:

Los casos de prueba fallidos para esta versión fueron 7, los mismos que son $T_f = (12, 13, 48, 55, 167, 17, 185)$ del conjunto de 207 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 140 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 19. Registro de la prueba de regresión de la versión 6 de Silabo

seed	K	T	T _n	T'	T _f	T' _f	TR	R	P	RC
1	6	207	49	137	7	7	33.8%	100%	5%	0.10
10	5	207	49	174	7	7	15.9%	100%	4%	0.08
50	3	207	49	178	7	7	14.0%	100%	4%	0.08
100	9	207	49	35	7	7	83.1%	100%	20%	0.33
150	6	207	49	139	7	7	32.9%	100%	5%	0.10
200	5	207	49	161	7	7	22.2%	100%	4%	0.08
250	6	207	49	130	7	7	37.2%	100%	5%	0.10
500	5	207	49	178	7	7	14.0%	100%	4%	0.08
750	5	207	49	198	7	7	4.3%	100%	4%	0.07
1000	7	207	49	78	7	7	62.3%	100%	9%	0.16

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 20. Comparativo de las técnicas en la versión 6 de Silabo

Cuadro comparativo de resultados de aplicación de los métodos en la versión 6 de Silabo		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	32%	0%
Recuperación (Recall)	100%	100%
Precision	6%	5%
Rendimiento combinado	0.12	0,06

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.6, con una capacidad de reducción de casos de prueba del 32%.

Versión 7:

Los casos de prueba fallidos para esta versión fueron 9, los mismos que son $T_f = (57, 59, 68, 73, 140, 144, 150, 153, 167)$ del conjunto de 239 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 110 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 21. Registro de la prueba de regresión de la versión 7 de Silabo

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	9	239	32	31	9	9	87.0%	100%	29%	0.45
10	7	239	32	180	9	9	24.7%	100%	5%	0.10
50	3	239	32	195	9	9	18.4%	100%	5%	0.09
100	9	239	32	31	9	9	87.0%	100%	29%	0.45
150	4	239	32	40	9	9	83.3%	100%	23%	0.37
200	7	239	32	170	9	9	28.9%	100%	5%	0.10
250	5	239	32	41	9	9	82.8%	100%	22%	0.36
500	5	239	32	40	9	9	83.3%	100%	23%	0.37
750	7	239	32	171	9	9	28.5%	100%	5%	0.10
1000	6	239	32	203	9	9	15.1%	100%	4%	0.08

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 22 Comparativo de las técnicas en la versión 7 de Silabo

Cuadro comparativo de resultados de aplicación de los métodos en la versión 7 de Silabo		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	54%	0%
Recuperación (Recall)	100%	100%
Precision	15%	5%
Rendimiento combinado	0.25	0,05

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.20, con una capacidad de reducción de casos de prueba del 54%.

Versión 8:

Los casos de prueba fallidos para esta versión fueron 5, los mismos que son $T_f = (11, 12, 13, 120, 207)$ del conjunto de 261 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 6, los mismos que agruparon en un promedio de 186 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 23. Registro de la prueba de regresión de la versión 8 de Silabo

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	3	261	22	215	5	5	17.6%	100%	2%	0.05
10	3	261	22	213	5	5	18.4%	100%	2%	0.05
50	7	261	22	205	5	5	21.5%	100%	2%	0.05
100	7	261	22	16	5	5	93.9%	100%	31%	0.48
150	2	261	22	227	5	5	13.0%	100%	2%	0.04
200	3	261	22	213	5	5	18.4%	100%	2%	0.05
250	4	261	22	178	5	5	31.8%	100%	3%	0.05
500	4	261	22	208	5	5	20.3%	100%	2%	0.05
750	5	261	22	213	5	5	18.4%	100%	2%	0.05
1000	4	261	22	174	5	5	33.3%	100%	3%	0.06

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 24. Comparativo de las técnicas en la versión 8 de Silabo

Cuadro comparativo de resultados de aplicación de los métodos en la versión 8 de Silabo		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	29%	0%
Recuperación (Recall)	100%	100%
Precision	5%	5%
Rendimiento combinado	0.09	0,02

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.07, con una capacidad de reducción de casos del 29%.

Versión 9:

Los casos de prueba fallidos para esta versión fueron 4, los mismos que son $T_f = (10, 12, 14, 168)$ del conjunto de 300 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 189 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 25. Registro de la prueba de regresión de la versión 9 de Silabo

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	3	300	39	284	4	4	5.3%	100%	1%	0.03
10	7	300	39	180	4	4	40.0%	100%	2%	0.04
50	3	300	39	195	4	4	35.0%	100%	2%	0.04
100	4	300	39	10	4	4	96.7%	100%	40%	0.57
150	4	300	39	238	4	4	20.7%	100%	2%	0.03
200	7	300	39	170	4	4	43.3%	100%	2%	0.05
250	4	300	39	219	4	4	27.0%	100%	2%	0.04
500	3	300	39	191	4	4	36.3%	100%	2%	0.04
750	4	300	39	204	4	4	32.0%	100%	2%	0.04
1000	6	300	39	203	4	4	32.3%	100%	2%	0.04

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 26. Comparativo de las técnicas en la versión 9 de Silabo

Cuadro comparativo de resultados de aplicación de los métodos en la versión 9 de Silabo		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	37%	0%
Recuperación (Recall)	100%	100%
Precision	6%	5%
Rendimiento combinado	0.09	0,01

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.08, con una capacidad de reducción de casos de prueba del 37%.

b. Impacto Global de los fallos en desarrollo del producto Silabo

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante las técnicas consideradas, en las diferentes versiones de Silabo, se obtiene el siguiente cuadro:

Tabla 27. Comparativo global de las técnicas en el producto software Silabo

Cuadro comparativo de resultados de aplicación de los métodos en el producto de software Silabo		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	40%	0%
Recuperación (Recall)	100%	100%
Precision	12%	6%
Rendimiento combinado	0.20	0,09

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización, con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima al valor recomendado de 1 del rendimiento combinado, considerando la capacidad de reducción de casos de prueba del 40%.

Producto Software Académico

Versión 2:

Los casos de prueba fallidos para esta versión fueron 6, los mismos que son $T_f = (8, 10, 71, 74, 86, 90)$ del conjunto de 92 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 70 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 28 Registro de la prueba de regresión de la versión 2 de Académico

seed	K	T	T _n	T'	T _f	T' _f	TR	R	P	RC
1	3	92	47	82	6	6	10.9%	100%	7%	0.14
10	3	92	47	82	6	6	10.9%	100%	7%	0.14
50	3	92	47	82	6	6	10.9%	100%	7%	0.14
100	4	92	47	15	6	6	83.7%	100%	40%	0.57
150	7	92	47	67	6	6	27.2%	100%	9%	0.16
200	4	92	47	80	6	6	13.0%	100%	8%	0.14
250	5	92	47	82	6	6	10.9%	100%	7%	0.14
500	3	92	47	80	6	6	13.0%	100%	8%	0.14
750	6	92	47	70	6	6	23.9%	100%	9%	0.16
1000	10	92	47	67	6	6	27.2%	100%	9%	0.16

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 29. Comparativo de las técnicas en la versión 2 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 2 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	23%	0%
Recuperación (Recall)	100%	100%
Precision	11%	6%
Rendimiento combinado	0.18	0,11

Lo cual implica una mejora del método propuesto versus el método de regresión total del 0.7, con una capacidad de reducción de casos de prueba del 23%.

Versión 3:

Los casos de prueba fallidos para esta versión fueron 5, los mismos que son $T_f = (63, 100, 133, 138, 159)$ del conjunto de 164 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 6, los mismos que agruparon en un promedio de 131 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 30. Registro de la prueba de regresión de la versión 3 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	7	164	72	126	5	5	23.2%	100%	4%	0.08
10	5	164	72	141	5	5	14.0%	100%	4%	0.07
50	6	164	72	164	5	5	0.0%	100%	3%	0.06
100	5	164	72	92	5	5	43.9%	100%	5%	0.10
150	5	164	72	128	5	5	22.0%	100%	4%	0.08
200	5	164	72	148	5	5	9.8%	100%	3%	0.07
250	8	164	72	123	5	5	25.0%	100%	4%	0.08
500	8	164	72	140	5	5	14.6%	100%	4%	0.07
750	4	164	72	102	5	5	37.8%	100%	5%	0.09
1000	5	164	72	153	5	5	6.7%	100%	3%	0.06

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 31. Comparativo de las técnicas en la versión 3 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 3 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	20%	0%
Recuperación (Recall)	100%	100%
Precision	3%	3%
Rendimiento combinado	0.07	0,07

Lo cual implica una no mejora del método propuesto versus el método de regresión total, con una capacidad de reducción de casos de prueba del 20%.

Versión 4:

Los casos de prueba fallidos para esta versión fueron 6, los mismos que son $T_f = (5, 61, 105, 150, 165, 200)$ del conjunto de 205 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 3, los mismos que agruparon en un promedio de 181 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 32. Registro de la prueba de regresión de la versión 4 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	2	205	41	205	6	6	0.0%	100%	3%	0.06
10	2	205	41	205	6	6	0.0%	100%	3%	0.06
50	2	205	41	205	6	6	0.0%	100%	3%	0.06
100	6	205	41	168	6	6	18.0%	100%	4%	0.07
150	2	205	41	205	6	6	0.0%	100%	3%	0.06
200	3	205	41	58	6	6	71.7%	100%	10%	0.19
250	2	205	41	205	6	6	0.0%	100%	3%	0.06
500	3	205	41	181	6	6	11.7%	100%	3%	0.06
750	3	205	41	205	6	6	0.0%	100%	3%	0.06
1000	7	205	41	179	6	6	12.7%	100%	3%	0.06

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 33. Comparativo de las técnicas en la versión 4 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 4 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	11%	0%
Recuperación (Recall)	100%	100%
Precision	3%	2%
Rendimiento combinado	0.07	0,05

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0,02 y una capacidad de reducción de casos de prueba del 11%.

Versión 5:

Los casos de prueba fallidos para esta versión fueron 4, los mismos que son $T_f = (133, 138, 155, 159)$ del conjunto de 245 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 4, los mismos que agruparon en un promedio de 61 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 34. Registro de la prueba de regresión de la versión 5 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	4	245	40	131	4	4	46.5%	100%	3%	0.06
10	3	245	40	12	4	4	95.1%	100%	33%	0.50
50	5	245	40	45	4	4	81.6%	100%	9%	0.16
100	4	245	40	11	4	4	95.5%	100%	36%	0.53
150	3	245	40	121	4	4	50.6%	100%	3%	0.06
200	5	245	40	45	4	4	81.6%	100%	9%	0.16
250	4	245	40	121	4	4	50.6%	100%	3%	0.06
500	2	245	40	11	4	4	95.5%	100%	36%	0.53
750	3	245	40	25	4	4	89.8%	100%	16%	0.28
1000	4	245	40	89	4	4	63.7%	100%	4%	0.09

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 35. Comparativo de las técnicas en la versión 5 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 5 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	8%	0%
Recuperación (Recall)	100%	100%
Precision	15%	1%
Rendimiento combinado	0.24	0,19

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.03 y con una capacidad de reducción de casos de prueba del 8%.

Versión 6:

Los casos de prueba fallidos para esta versión fueron 6, los mismos que son Tf = (27, 78, 150, 190, 224, 290) del conjunto de 302 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 4, los

mismos que agruparon en un promedio de 254 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 36. Registro de la prueba de regresión de la versión 6 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	7	302	57	192	6	6	36.4%	100%	3%	0.06
10	3	302	57	290	6	6	4.0%	100%	2%	0.04
50	2	302	57	302	6	6	0.0%	100%	2%	0.04
100	3	302	57	269	6	6	10.9%	100%	2%	0.04
150	4	302	57	76	6	6	74.8%	100%	8%	0.15
200	4	302	57	274	6	6	9.3%	100%	2%	0.04
250	3	302	57	295	6	6	2.3%	100%	2%	0.04
500	5	302	57	263	6	6	12.9%	100%	2%	0.04
750	5	302	57	299	6	6	1.0%	100%	2%	0.04
1000	7	302	57	282	6	6	6.6%	100%	2%	0.04

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla37. Comparativo de las técnicas en la versión 6 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 6 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	15%	0%
Recuperación (Recall)	100%	100%
Precision	2%	1%
Rendimiento combinado	0.5	0,01

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.04 y con una capacidad de reducción de casos de prueba del 15%.

Versión 7:

Los casos de prueba fallidos para esta versión fueron 4, los mismos que son $T_f = (22, 180, 210, 354)$ del conjunto de 356 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 272 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 38. Registro de la prueba de regresión de la versión 7 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	4	356	54	331	4	4	7.0%	100%	1%	0.02
10	5	356	54	87	4	4	75.6%	100%	5%	0.09
50	7	356	54	302	4	4	15.2%	100%	1%	0.03
100	4	356	54	334	4	4	6.2%	100%	1%	0.02
150	7	356	54	296	4	4	16.9%	100%	1%	0.03
200	6	356	54	317	4	4	11.0%	100%	1%	0.02
250	5	356	54	315	4	4	11.5%	100%	1%	0.03
500	4	356	54	181	4	4	49.2%	100%	2%	0.04
750	3	356	54	334	4	4	6.2%	100%	1%	0.02
1000	7	356	54	225	4	4	36.8%	100%	2%	0.03

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 39. Comparativo de las técnicas en la versión 7 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 7 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	23%	0%
Recuperación (Recall)	100%	100%
Precision	2%	1%
Rendimiento combinado	0.04	0,01

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.03 y con una capacidad de reducción de casos de prueba del 23%.

Versión 8:

Los casos de prueba fallidos para esta versión fueron 6, los mismos que son $T_f = (133, 156, 198, 230, 280, 362)$ del conjunto de 406 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 195 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 40. Registro de la prueba de regresión de la versión 8 de Académico

seed	K	T	Tn	T'	T _f	T' _f	TR	R	P	RC
1	6	406	50	350	6	6	13.8%	100%	2%	0.03
10	5	406	50	335	6	6	17.5%	100%	2%	0.04
50	6	406	50	25	6	6	93.8%	100%	24%	0.39
100	4	406	50	382	6	6	5.9%	100%	2%	0.03
150	4	406	50	37	6	6	90.9%	100%	16%	0.28
200	6	406	50	353	6	6	13.1%	100%	2%	0.03
250	6	406	50	336	6	6	17.2%	100%	2%	0.04
500	4	406	50	66	6	6	83.7%	100%	9%	0.17
750	6	406	50	36	6	6	91.1%	100%	17%	0.29
1000	4	406	50	37	6	6	90.9%	100%	16%	0.28

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 41. Comparativo de las técnicas en la versión 8 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 8 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	51%	0%
Recuperación (Recall)	100%	100%
Precision	9%	1%
Rendimiento combinado	0.15	0,01

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.14 y con una capacidad de reducción de casos de prueba del 51%.

Versión 9:

Los casos de prueba fallidos para esta versión fueron 4, los mismos que son $T_f = (220, 280, 420, 430)$ del conjunto de 453 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 5, los mismos que agruparon en un promedio de 267 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 42. Registro de la prueba de regresión de la versión 9 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	3	453	47	219	4	4	51.7%	100%	2%	0.04
10	4	453	47	448	4	4	1.1%	100%	1%	0.02
50	7	453	47	418	4	4	7.7%	100%	1%	0.02
100	4	453	47	231	4	4	49.0%	100%	2%	0.03
150	4	453	47	47	4	4	89.6%	100%	9%	0.16
200	5	453	47	312	4	4	31.1%	100%	1%	0.03
250	6	453	47	406	4	4	10.4%	100%	1%	0.02
500	4	453	47	439	4	4	3.1%	100%	1%	0.02
750	6	453	47	108	4	4	76.2%	100%	4%	0.07
1000	4	453	47	46	4	4	89.8%	100%	9%	0.16

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 43. Comparativo de las técnicas en la versión 9 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 9 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	41%	0%
Recuperación (Recall)	100%	100%
Precision	3%	0%
Rendimiento combinado	0.05	0,00

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.05 y con una capacidad de reducción de casos de prueba del 41%.

Versión 10:

Los casos de prueba fallidos para esta versión fueron 3, los mismos que son $T_f = (363, 408, 432)$ del conjunto de 480 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de 4, los mismos que agruparon en un promedio de 179 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 44. Registro de la prueba de regresión de la versión 10 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	5	480	27	27	3	3	94.4%	100%	11%	0.20
10	4	480	27	363	3	3	24.4%	100%	1%	0.02
50	5	480	27	31	3	3	93.5%	100%	10%	0.18
100	7	480	27	31	3	3	93.5%	100%	10%	0.18
150	4	480	27	29	3	3	94.0%	100%	10%	0.19
200	4	480	27	211	3	3	56.0%	100%	1%	0.03
250	3	480	27	270	3	3	43.8%	100%	1%	0.02
500	3	480	27	298	3	3	37.9%	100%	1%	0.02
750	3	480	27	305	3	3	36.5%	100%	1%	0.02
1000	4	480	27	228	3	3	52.5%	100%	1%	0.03

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 45. Comparativo de las técnicas en la versión 10 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 10 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	62%	0%
Recuperación (Recall)	100%	100%
Precision	4%	0%
Rendimiento combinado	0.08	0,00

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.08 y con una capacidad de reducción del 62%.

Versión 11:

Los casos de prueba fallidos para esta versión fueron 5, los mismos que son $T_f = (190, 362, 380, 520, 188)$ del conjunto de 528 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres (K) de los mismos que agruparon en un promedio de 324 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 46. Registro de la prueba de regresión de la versión 11 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	6	528	48	22	5	5	95.8%	100%	23%	0.37
10	5	528	48	408	5	5	22.7%	100%	1%	0.02
50	6	528	48	25	5	5	95.3%	100%	20%	0.33
100	6	528	48	22	5	5	95.8%	100%	23%	0.37
150	4	528	48	475	5	5	10.0%	100%	1%	0.02
200	5	528	48	351	5	5	33.5%	100%	1%	0.03
250	3	528	48	467	5	5	11.6%	100%	1%	0.02
500	4	528	48	508	5	5	3.8%	100%	1%	0.02
750	3	528	48	467	5	5	11.6%	100%	1%	0.02
1000	3	528	48	503	5	5	4.7%	100%	1%	0.02

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 47. Comparativo de las técnicas en la versión 11 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 11 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	38%	0%
Recuperación (Recall)	100%	100%
Precision	7%	0%
Rendimiento combinado	0.12	0,00

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.12 y con una capacidad de reducción de casos de prueba del 38%.

Versión 12:

Los casos de prueba fallidos para esta versión fueron 4, los mismos que son $T_f = (10, 100, 195, 574)$ del conjunto de 600 casos de prueba acumulados (T) que se clusterizaron en un promedio de clústeres 4 (K) de los mismos que agruparon en un promedio de 410 (T') casos de prueba para la prueba de regresión de la versión.

Tabla 48. Registro de la prueba de regresión de la versión 12 de Académico

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	RC
1	4	600	72	338	4	4	43.7%	100%	1%	0.02
10	6	600	72	158	4	4	73.7%	100%	3%	0.05
50	3	600	72	555	4	4	7.5%	100%	1%	0.01
100	5	600	72	501	4	4	16.5%	100%	1%	0.02
150	5	600	72	240	4	4	60.0%	100%	2%	0.03
200	5	600	72	491	4	4	18.2%	100%	1%	0.02
250	3	600	72	418	4	4	30.3%	100%	1%	0.02
500	4	600	72	555	4	4	7.5%	100%	1%	0.01
750	4	600	72	299	4	4	50.2%	100%	1%	0.03
1000	4	600	72	545	4	4	9.2%	100%	1%	0.01

Fuente: Propia

Considerando las métricas obtenidas en la aplicación del método propuesto versus la ejecución del método de regresión total, tenemos que:

Tabla 49. Comparativo de las técnicas en la versión 12 de Académico

Cuadro comparativo de resultados de aplicación de los métodos en la versión 12 de Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	31%	0%
Recuperación (Recall)	100%	100%
Precision	1%	0%
Rendimiento combinado	0.2	0,00

Lo cual implica una mejora del método propuesto versus el método de regresión total con 0.2 y con una capacidad de reducción de casos de prueba del 31%.

c. Impacto Global de los fallos en desarrollo del producto Académico

Analizando las métricas obtenidas en la ejecución de las pruebas de regresión mediante las técnicas consideradas, en las diferentes versiones de Silabo, se obtiene el siguiente cuadro:

Tabla 50. Comparativo global de las técnicas en el producto software Académico

Cuadro comparativo de resultados de aplicación de los métodos en el producto de software Académico		
Indicadores	Método propuesto	Método regresión total
Cantidad de casos de prueba	36%	0%
Recuperación (Recall)	100%	100%
Precision	6%	1%
Rendimiento combinado	0.10	0,01

Se hace notorio que algunas métricas de algunas versiones son mejores que otras, básicamente por el resultado del proceso de clusterización, con lo que se concluye que la aplicación del método de prueba de regresión es mejor que la prueba de regresión total, ya que es el que más se aproxima al valor recomendado de 1 del rendimiento global, considerando la capacidad de reducción de casos de prueba del 36%.