MEJORA DE LA ESTRATEGIA DE TESTING EN UNA EMPRESA DE DESARROLLO DE SOFTWARE, ALINEADO CON ALGUNAS ÁREAS DE CMMI, USANDO LA METODOLOGÍA DMAIC.

PAULA ANDREA SALGADO TORO

UNIVERSIDAD AUTÓNOMA DE MANIZALES
FACULTAD DE INGENIERIA
MAESTRÍA EN GESTIÓN Y DESARROLLO DE PROYECTOS DE SOFTWARE
MANIZALES
2020

MEJORA DE LA ESTRATEGIA DE TESTING EN UNA EMPRESADE DESARROLLO DE SOFTWARE, ALINEADO CON ALGUNAS ÁREAS DE CMMI, USANDO LA METODOLOGÍA DMAIC

CALIDAD Y MÉTRICAS DE SOFTWARE

Autora

PAULA ANDREA SALGADO TORO

Proyecto de grado para optar por el título de magister en gestión y desarrollo de proyectos de software.

Tutor

M.SC SANDRA VICTORIA HURTADO GIL

UNIVERSIDAD AUTÓNOMA DE MANIZALES

FACULTAD DE INGENIERA DE SISTEMAS

MAESTRÍA EN GESTIÓN Y DESARROLLO DE PROYECTOS DE SOFTWARE

MANIZALES

2020

RESUMEN

Objetivo: Optimizar la estrategia de testing de una empresa de desarrollo de software, mediante la metodología DMAIC, alineado con las áreas PPQA, VAL y VER de CMMI-DEV para el mejoramiento de la calidad de los productos.

Metodología: Se realizó una investigación aplicada, la cual consistió en poner en práctica la teoría con el propósito de mejorar la estrategia de testing, distribuida en 5 fases, equivalentes con las fases de DMAIC.

Resultados: Se realizó el diagnostico en una empresa de desarrollo, encontrado el número y tipos de pruebas indispensables para una estrategia de testing en comparación con las que actualmente se realizan, identificando los principales problemas que se presentan en la ejecución de la misma.

Se diseñó una estrategia de *testing*, partiendo de conceptos teóricos y buenas prácticas, que permitiera disminuir el número de desperdicios que se presentan en la estrategia actual, y de esta forma generar para la organización un proceso óptimo y eficiente.

La evaluación del proceso permite conocer el estado actual y disminuir las deviaciones de las acciones durante la ejecución en cada proyecto, así como plantear una serie de métricas para la verificación y seguimiento de la estrategia.

Conclusiones: Todo proceso es objeto de mejora, siempre y cuando se conozca las prácticas ideales o estándares de calidad que permitan comparar las actividades que se realizan actualmente frente a una calidad esperada. La brecha de mejoramiento disminuye cuando se genera un plan o modelo nuevo derivado de un diagnóstico inicial que identifique las fortalezas y debilidades de la estrategia actual, para eliminar los desperdicios del proceso.

Es importante el diseño de una estrategia de testing escalonada, que permita poco a poco asegurar la calidad del producto, donde el registro de los resultados mediante métricas es el insumo primordial para el seguimiento y para la gestión del conocimiento organizacional.

Palabras Claves: CMMI, VER, VAL, PPQA, DMAIC, Estrategia de testing, Pruebas de software, Mejoramiento de procesos, Mejora continua.

ABSTRACT

Objective: Optimize the testing strategy of a software development company, using the DMAIC methodology, aligned with the PPQA, VAL and VER areas of CMMI-DEV for the improvement of product quality.

Results: The diagnosis was made in a development company, finding the number and types of tests essential for a testing strategy compared to those currently being carried out, identifying the main problems that arise in its execution.
A testing strategy was designed based on theoretical concepts and good practices that would reduce the number of wastes presented in the current strategy and thus generating an optimal and efficient process for the organization.
The evaluation of the process allows knowing the actual status and reducing deviations of the actions during the execution of each project, as well as proposing a series of metrics for the verification and monitoring of the strategy.

Conclusions: Every process is subject to improvement, as long as the ideal practices or quality standards that allow comparing the activities currently carried out against an expected quality are known. The improvement gap decreases when a new derived plan or model is generated of an initial diagnosis that identifies the strengths and weaknesses of the current strategy, to eliminate waste from the process. It is important to design a staggered testing strategy that will gradually ensure product quality where the recording of results through metrics is the primary input for monitoring and for the management of organizational knowledge.

Key Words: CMMI, VER, VAL, PPQA, DMAIC, Testing strategy, Software testing, Process improvement, Continuous improvement.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

1 INTRODUCCIÓN

En la actualidad el software hace parte integral e indispensable en la innovación de los servicios y productos que son usados a diario, un ejemplo de ello son los celulares, vehículos, aplicaciones móviles, entre otros; los cuales son creados por el ser humano para dar respuesta a las necesidades del medio y del mercado. Éste, al ser desarrollado por personas, puede presentar fallas, que si no son detectadas a tiempo se convierten en un riesgo latente y pueden generar daños en los recursos físicos, humanos, naturales o financieros, así como reprocesos y aumentos de tiempos en las entregas.

Muchas veces durante la ejecución de un proyecto las prioridades del equipo desarrollador se ven afectadas debido a que el cliente reporta un error posterior a su entrega, el cual debe ser solucionado a la mayor brevedad posible, lo que implica parar y volcar los esfuerzos en dicha solución. A lo que el equipo se pregunta ¿Cómo fue posible que se presentaran dichos errores? Expuesto lo anterior es de gran importancia para las empresas de desarrollo de software generar productos de calidad, donde los errores no sean encontrados por el cliente en producción, sino corregidos previamente a su entrega y es aquí donde la estrategia de testing toma valor y comienza a ser parte integral del ciclo de vida del desarrollo de un producto.

Si bien las empresas tienen definidos sus procesos y estrategias, las exigencias del mercado, las obliga a pensar en un mejoramiento continuo, o de lo contrario cabe la posibilidad de su desaparición y liquidación. Por consiguiente, el mejoramiento las hace más visibles y les permite tener un sello propio para sobresalir en un mercado competitivo.

El siguiente trabajo de grado muestra una investigación aplicada donde se combinan dos metodologías internacionales las cuales dan como resultado un diagnóstico y una propuesta para la mejora de la estrategia de testing y disminución de los desperdicios actuales en una empresa de desarrollo.

2 ÁREA PROBLEMÁTICA Y PREGUNTA DE INVESTIGACIÓN

2.1 ANTECEDENTES

Existen diferentes autores que han explorado la mejora de procesos en diferentes industrias mediante el uso de una o múltiples metodologías, volcando esfuerzos en el diagnóstico institucional y planeamiento de acciones de mejoramiento interno. Es el caso del proyecto expuesto por Iza Chorlango, el cual implementó la metodología Six Sigma como modelo para el mejoramiento de la calidad durante el proceso de desarrollo de software, alcanzando el objetivo del aumento en la satisfacción del cliente, la reducción y eliminación de defectos (Iza Chorlango, 2012) (Bueno, Crespo, & Jino, 2006).

Six Sigma es una metodología vanguardista e innovadora utilizada en sectores industriales y tecnológicos, aplicada con éxito para la excelencia en el rendimiento de los procesos. Un estudio realizado por Shirshendu Roy en el 2015 (Shirshendu & Sujoy , 2015) demostró que el desempeño de la organización mejoró al reducir significativamente los reprocesos desperdicios con la aplicación de esta metodología. Por la misma línea de investigación, Brucker y Julliand en el 2014, en su caso de estudio utilizaron herramientas como el histograma, gráfico de Pareto y modelo de causa y efecto, para reducir los defectos y mejorar la calidad del software (Brucker & Julliand, 2014). Para este trabajo de investigación se utilizaron algunas de las herramientas mencionadas anteriormente, con el objetivo de verificar su aporte al momento de realizar mejoras en los procesos.

La metodología DMAIC ha sido aplicada en diferentes estudios y organizaciones para evaluar, conocer y eliminar los elementos del proyecto que incrementa los tiempos de finalización del mismo, mejorando de esta forma la satisfacción del cliente al obtener exactamente lo requerido en menor tiempo y con menor tasa de defectos. (Chauhan, 2015) (Salazar Moreno, Mendoza Muñoz, Montoya Reyes, & Castañeda, 2018).

Otro estudio propuso una plantilla en XML para la generación directa de historias de usuario, dentro de una metodología ágil, facilitando el control de versiones y creación de

esquemas preconceptuales ejecutables, donde permite la interacción dinámica entre el interesado (cliente) y los analistas de requerimientos, obteniendo un mayor nivel de detalle para la ejecución de las pruebas (Martín Córdoba, 2015) (Villamizar Suaza, Tabares García, & Zapata Jaramillo, 2015). Plantillas como las propuestas en este estudio, pueden servir como base para la creación de herramientas que permitan un mayor detalle y control de las pruebas.

Leaños Rodríguez, en su caso de estudio, identificó los problemas existentes en el proceso de desarrollo de software de su compañía, y mediante el uso de la matriz de Vester buscó la asociación entre los distintos problemas de los proyectos e identificó cuales eran pasivos, críticos, indiferente y activos; asimismo, demostró que con el uso de pruebas estadísticas se podía realizar el seguimiento de la curva de mejoramiento del proceso en el proyecto a través del tiempo. (Leaños Rodriguez, 2018).

Autores como Chiuki, Rubinstein y colaborares en el 2015 utilizaron una combinación de modelos para el mejoramiento de la calidad, integrando CMMI y MPS-SW donde se destacó que estas combinaciones de modelos permiten analizar en 360 grados los procesos internos de las empresas, alcanzando objetivos de manera eficaz y posicionándolas en el mercado. Por otra parte Delgado Dapena en el 2014 realizó un estudio para el mejoramiento del proceso de software, partiendo del diagnóstico y planteamiento de factores para el éxito, al realizar una evaluación crítica del modelo IDEAL y el modelo Ágil SPI; los resultados arrojados mostraron las barreras existentes para el mejoramiento del procesos de software, la experiencia del equipo de desarrollo y los años en el rol que desempeñan son la principal causa a intervenir para el mejoramiento de un proceso (Chiuki, y otros, 2015) (Chanatasing Toapanta, Oña Sinchiguano, & Orbea Jimenez, 2017).

Autores como Magallanes de los Ríos y Vega Marca realizaron un proyecto utilizando el modelo CMMI donde recopilaron información para dar como resultado herramientas de medición para la productividad del software, del personal y de las herramientas de gestión de riesgos (Magallanes de los rios & Vega Marca, 2015); esta referencia sirve como aporte

para tener en cuenta las herramientas utilizadas para la generación de indicadores que generen el valor requerido por la compañía.

Flore Montes y Llamoca Quispe, usaron los modelos IDEAL y CMMI-DEV, para la mejora de los indicadores de rendimiento, en la planificación, monitorización y control de proyectos, así como en la gestión de los requisitos. Srinivasan y colaboradores muestran que el uso de metodologías como DMAIC, acompañada con otros enfoques, ayuda a descubrir mayores causas, para eliminar la variación de los procesos. El uso de los aspectos de medición de defectos Sigma, son útiles para mejorar el diseño de modelos de estimación de defectos de software, gracias al uso de tablas y diagramas de respuestas, para conseguir el nivel óptimo de un producto (Flores Montes & LLamoca Quispe, 2017) (Srinivasan, Muthu, Prasad, & Satheesh, 2014) (Mhammed Almakadmeh, 2017).

Tanto DMAIC (Six Sigmas) como CMMI-DEV implican el uso de métodos cualitativos y cuantitativos, lo que permite una mejora en la eficiencia de los procesos de desarrollo de software, siempre y cuando sean soportados en diferentes herramientas; en la literatura se han propuesto metamodelos para la validación del software que dan facilidad a las empresas de desarrollo para identificar, preparar y validar las pruebas de un proyecto de acuerdo a sus especificaciones, Jiménez Puello y colaboradores obtuvieron como resultado un metamodelo que proporciona técnicas, herramientas y criterios que dan mayor fiabilidad y aprendizaje con lo relacionado a la reducción de errores durante el desarrollo del producto, aumentando de esta forma la satisfacción del cliente (Jimenez Puello, San feliu, & Calvo Manzano, 2016).

Dentro del proceso específico de testing, el uso de buenas prácticas conduce a un proceso más eficiente (dentro del presupuesto y el cronograma) y más efectivo (menos errores implementados en producción) elementos cruciales para el éxito del desarrollo de software. Sanz, Saldaña y García implementaron el modelo TestPAI (Sanz, Saldaña, Garcia, & Gaitero, 2009) integrando satisfactoriamente algunas áreas de CMMI-DEV, lo cual les permitió rediseñar el proceso de testing y crear nuevos subprocesos para la mejora

organizacional. Por la misma línea, Albarracín, Correa y colaboradores presentan una mejora de proceso conformado por un conjunto de prácticas, actividades, roles y productos esenciales para la gestión cuantitativa (Sanz, Saldaña, Garcia, & Gaitero, 2009) (Ardila Albarracín, Pino Correa, Pardo Calvache, & Merchán Paredes, 2014).

Rojas Montes y colaboradores propusieron un proceso de pruebas enfocado a pequeñas empresas de desarrollo de software, mediante una guía que involucra pruebas funcionales, indicando ¿qué hacer? y ¿cómo hacerlo?, donde lograron demostrar en campo que la estandarización del proceso de pruebas mediante el uso de 6 fases (Iniciación, planeación, diseño, ejecución, monitoreo y control, y finalización) mejora los tiempo y las entregas de los módulos del producto, disminuyendo el esfuerzo (días/horas) utilizado en cada ciclo (Rojas Montes, Pino Correa, & Maur, 2015) .

Dapozzo y colaboradores propusieron una serie de pruebas en cuatro niveles (prueba inicial, de sistema, de aceptación y de exploración); esta mejora al proceso permitió controlar y unificar la información, asimismo fue posible detectar y dar seguimiento a los defectos; se concluyó que a mayor especificidad del método, menos complejo es la realización de los ajustes, a mayor disponibilidad y pertinencia de datos históricos del proyecto, mejor es la precisión de la estimación (Dapozzo, y otros, 2016).

Por último, las investigaciones manifiestan que el número de fallas no depende solo de la elección de una metodología correcta de pruebas y la estandarización de la misma, sino la intervención desde el principio del desarrollo de software, por tal razón es indispensable utilizar combinaciones de metodologías de pruebas que garanticen robustez y completitud de requisitos del cliente (Alvarez Baez, 2017).

2.2 ÁREA PROBLEMÁTICA

En la actualidad, las industrias entregan servicios y productos de manera más rápida, aumentando el número de empresas competitivas en todos los sectores, por ello es indispensable para el sostenimiento y prestigio, incluir dentro de los lineamientos organizacionales el mejoramiento de la calidad; exigencias *per sé* por parte del medio y del cliente final, el cual castiga o penaliza los contratos por el retraso de los proyectos o errores en los mismos, la demanda que existe en el medio lleva a que las empresas deban tener mayor control de sus procesos para obtener mejores resultados.

En la industria de la tecnología los cambios son mucho más rápidos, razón por la cual el mejoramiento de la calidad debe estar presente en todos sus procesos; no obstante, lo que se observa es que el cumplimiento de tiempos, la detección y solución de los errores siguen siendo las falencias más comunes en este sector, debido a una gestión deficiente, alcances no definidos, mala planeación o baja calidad del producto. También influye que la mayoría de las veces los tiempos de respuesta son determinados en plazos que el cliente propone y sobre estos no se tienen en cuenta modificaciones al alcance, procesos de formación del equipo desarrollador, tiempo en la ejecución de pruebas y corrección de defectos encontrados, ni otros factores que puedan afectar la entrega (Ardila Albarracín, Pino Correa, Pardo Calvache, & Merchán Paredes, 2014).

Una forma de contribuir al mejoramiento de la calidad es mediante la aplicación de modelos genéricos que recopilan buenas prácticas y están reconocidos a nivel internacional. Sin embargo, las pequeñas empresas desarrolladoras de productos de software tienen dificultades para la adopción de estas buenas prácticas de gestión de procesos de calidad ofrecidas por diferentes modelos, enfoques o metodologías como DMAIC[1], CMMI[2],

[1] Definir, Medir, Analizar, Mejorar, Controlar, por sus siglas en inglés

[2] *Capability Maturity Model Integration*: Modelo de Madurez de Capacidad Integrado

ISO/IEC 12207[3], Modelo IDEAL[4], entre otros, debido principalmente a los costos e inversiones a largo plazo. Sin embargo, las empresas no podrán obtener mayor rendimiento, ni fiabilidad del producto si no tienen control de la variabilidad del proceso (Ardila Albarracín, Pino Correa, Pardo Calvache, & Merchán Paredes, 2014) (Sokovic, Pavletic, & Kern Pipan, 2010).

La baja fiabilidad de los productos genera mayores requerimientos de mantenimiento: Estudios han demostrado que en las empresas de desarrollo el 97% tienen que tratar con defectos un año después de ser entregados y el 55% siguen solucionando problemas 2 años después de la implementación (Jimenez Puello, San feliu, & Calvo Manzano, 2016); por ende toda empresa de tecnología o donde se desarrolle software debe contar con mecanismos de control para verificar que el producto cuente con las normas, proceso y estándares altos de calidad del medio.

Modelos como los mencionados previamente entregan buenas prácticas en el proceso de verificación y validación, dado que para lograr la calidad del proceso de software es necesario tener calidad en las pruebas que se utilicen; de esta manera, la solución a la mayoría de los errores identificados en la estrategia de testing, se corrigen con un proceso definido, gestionado y controlado (Sanz, Saldaña, Garcia, & Gaitero, 2009).

Actualmente la empresa Ceiba Software pertenece al selecto grupo a nivel nacional que se encuentra certificado en CMMI-DEV nivel 5, este modelo reúne una serie de mejores prácticas que ayuda a la organización a perfeccionar la efectividad, eficiencia y calidad de sus procesos, lo que genera nuevos retos en el mantenimiento de éste e implica una gestión continua hacia la mejora de los procesos. Lo anterior dentro del marco de una metodología ágil.

[3] Estándar para los procesos del ciclo de vida del software

[4] Inicio, Diagnóstico, Establecimiento, Actuación y Aprendizaje, por sus siglas en inglés

A pesar de lo anterior, la organización no tiene métricas trazables que permitan clasificar la fuente de los errores, la fase del proceso de desarrollo en la que se generó y el tiempo de corrección de defectos. De igual manera no se identifican qué actividades realizadas dentro del proceso de testing aportan valor a los indicadores para el desarrollo de los productos, siendo este proceso, al igual que los demás, susceptible a mejoras.

2.3 PREGUNTA DE INVESTIGACIÓN

¿Cómo implementar con DMAIC una mejora en la estrategia de testing de la empresa Ceiba Software, alineándose con las áreas PPQA, VAL y VER de CMMI-DEV, para aumentar la calidad del software producido?

3 JUSTIFICACIÓN

Partiendo del contexto de la Maestría en Gestión y Desarrollo de Proyectos de Software, este trabajo tiene un gran vínculo con el objeto de ésta, siendo una propuesta pertinente para el mejoramiento de la calidad de los productos de software y el aporte a nuevo conocimiento empírico a partir de fundamentos teóricos, útil en la gestión a la hora de tomar decisiones bien sea dentro del proyecto o en la mejora del proceso de software.

El trabajo de investigación propuesto es novedoso al partir de la sinergia entre las áreas de proceso de CMMI-DEV relacionadas con pruebas (VAR, VEL, PPQA) y DMAIC, teniendo en cuenta la poca cantidad de publicaciones encontradas donde se relacionan ambas metodologías, y más aún para procesos de testing.

Los resultados obtenidos proporcionarán datos valiosos para el mejoramiento y conservación de la certificación CMMI-DEV nivel 5 de la empresa, a su vez y en mayor proporción al área objeto de intervención (testing), mejorando la alineación con los objetivos del negocio, la eficacia en la detección de errores a lo largo del ciclo de vida de los proyectos, reducción en el número de errores que afectan directamente a los clientes, logrando un fin último en la mejora de la calidad del producto e indirectamente en la satisfacción del cliente y disminución de costos por reprocesos o desperdicios.

Además, el marco teórico y metodológico propuesto para el mejoramiento de la estrategia de Testing con base al modelo CMMI-DEV y DMAIC puede servir de cimiento para trabajos investigativos futuros dentro de la academia y para empresas en el campo del desarrollo de software.

4 REFERENTE TEÓRICO

4.1 MARCO CONCEPTUAL

4.1.1 CMMI-DEV

CMMI-DEV por sus siglas en inglés (Capability Maturity Model Integration for Development) es un modelo de referencia que cubre actividades propias del desarrollo de productos y servicios para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software. Este es un modelo genérico de calidad basado en procesos, que ayuda a las empresas a optimizar los mismos, adquiriendo un valor mayor de competitividad y eficiencia, y contribuyendo para internacionalizar los productos y servicios de software (Puello, 2013) (Software Engineering Institute, 2010) (Esterkin & Pons, 2012).

CMMI-DEV establece cinco niveles de madurez en función de si la organización tiene o no una serie de características específicas, cada nivel proporciona una capa en la base de mejoramiento continuo del proceso. La evaluación de estas especificaciones puede otorgar el nivel de madurez, de acuerdo con un modelo de buenas prácticas.

Para alcanzar cada nivel de madurez es necesario cumplir con unas áreas de proceso específicas. El modelo CMMI-DEV-v1.3 define 22 áreas de proceso, categorizadas según cuatro grupos: Gestión de procesos, Gestión de proyectos, Ingeniería y Soporte, que conforman un conjunto de buenas prácticas en las actividades relacionadas con un determinado proceso del desarrollo del software (Puello, 2013) (Software Engineering Institute, 2010).

La Tabla 1 muestra las áreas de CMMI-DEV-v1.3 integradas en cada uno de los grupos. Con relación al proceso de pruebas se definen tres áreas de proceso que se trabajaron en este proyecto: Aseguramiento de la Calidad del Proceso y del Producto (PPQA), Validación (VAL) y Verificación (VER). Estas áreas se seleccionaron por su estrecha relación la estrategia, puesto que buscan que los requisitos y el producto final estén dentro de los

estándares ideales que la organización define (Puello, 2013) (Sanz, Saldaña, Garcia, & Gaitero, 2009).

Tabla 1. Categorías de procesos según CMMI-Dev

GESTIÓN DE PROCESOS	GESTIÓN DE PROYECTOS	INGENIERÍA	SOPORTE
Definición de procesos de la organización (ODP)	Gestión Integrada del Proyecto (IPM).	Desarrollo de Requisitos (RD).	Análisis Causal y Resolución (CAR).
Enfoque en procesos de la organización (OPF)	Monitorización y Control del Proyecto (PMC).	Solución Técnica (TS).	Gestión de Configuración (CM).
Gestión del rendimiento de la organización (OPM)	Planificación del Proyecto (PP).	Validación (VAL).	Análisis de Decisiones y Resolución (DAR).
Formación en la organización (OT)	Gestión Cuantitativa del Proyecto (QPM).	Verificación (VER).	Medición y Análisis (MA).
	Gestión de Requisitos (REQM).	Desarrollo de Requisitos (RD).	Aseguramiento de la Calidad del Proceso y del Producto (PPQA)
	Gestión de Riesgos (RSKM).	Integración del Producto (PI).	
	Gestión de Acuerdos con Proveedores (SAM).		

Fuente: Elaboración propia.

Área Aseguramiento de la Calidad del Proceso y del Producto (PPQA):

El área tiene como propósito evaluar de manera objetiva la adherencia a los procesos definidos en la organización, asegurándose de la resolución de las no conformidades detectadas, al mismo tiempo llevar un registro de los resultados obtenidos e informarlos

(Trujillo Casañola, Febles Estrada, Garcia Rodriguez, Betancourt Rodriguez, & Cao Terrero, 2011).

Verificación (VER)

VER permite evaluar el sistema o sus componentes, identificando defectos en etapas tempranas de la creación y en fases determinadas del producto y reduciendo los altos costos asociados a la identificación y corrección de defectos que se pueden presentar más adelante, causante de disminución en los tiempos de la entrega y costos (Tigueros Vargas, 2015). La verificación responde a la pregunta ¿Se está construyendo el producto correctamente? (Gonzalez Palacio, 2009).

Esta área tiene como propósito que el producto internamente desarrollado cumpla con las especificaciones de requisitos, utiliza métodos como la inspección, revisión por pares, auditorias, análisis, simulaciones, pruebas y demostraciones.

Validación (VAL):

Las prácticas de VAL demuestran que el producto fabricado puede ser utilizado y se ajusta a su uso previsto cuando se sitúa en su entorno previsto. Estas actividades son aplicables a todos los aspectos del producto en cualquiera de sus entornos, tales como operación, formación, fabricación, mantenimiento y servicios de soporte; un buen nivel de VAL repercute en la satisfacción del cliente debido a que evalúa el producto según los requisitos especificados (Tigueros Vargas, 2015) (Mera Paz J. A., 2016) (IEEE, ISO / IEC / IEEE 24765:2010, 2010)

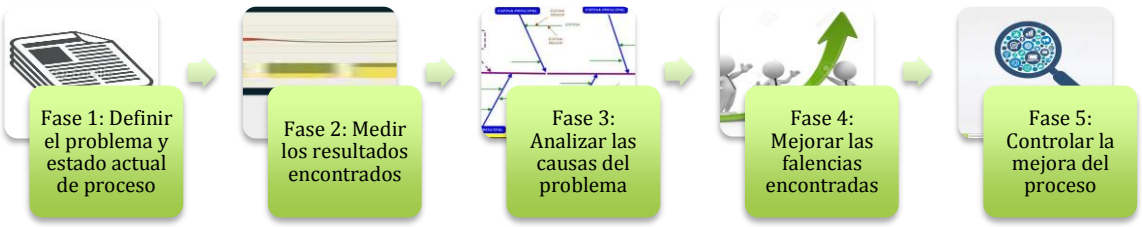
Este conjunto de actividades debe dar respuesta a la pregunta ¿Se está construyendo el producto correcto? (Gonzalez Palacio, 2009), respuesta dada al evaluar los requisitos, diseño y prototipo del producto, que satisfacen las necesidades del usuario y sus expectativas.

4.1.2 DMAIC

DMAIC es el modelo de mejora que utiliza la metodología Six Sigma. Es un ciclo de mejoramiento de proceso basado en hechos (datos), formado por el acrónimo de sus fases interconectadas en inglés: Definir, Medir, Analizar, Mejorar y Controlar (*Define*, *Measure*, *Analyse*, *Improve*, *Control*). Parte del principio según el cual, si la organización no puede definir, no se puede medir y por lo tanto no se puede mejorar y sostener la calidad logrando una eliminación de pasos improductivos o desperdicios (Sokovic, Pavletic, & Kern Pipan, 2010).

Cada una de las fases de DMAIC (Ilustración 1) utiliza herramientas para dar respuestas a ciertas preguntas específicas del proceso (Ocampo & Pavon, 2012), como se explicará a continuación.

Ilustración 1. Modelo DMAIC



Fuente: Elaboración Propia

Fase 1 - Definir:

Comprende la etapa de identificación, priorización y selección del proyecto, mediante una planeación que involucra las expectativas (metas y objetivos) y necesidades de los clientes

(requisitos o historias de usuario[5]), la identificación del proceso y variables críticas siendo la fase encaminada al diagnóstico preliminar de la organización y alcance del proyecto (Sokovic, Pavletic, & Kern Pipan, 2010) (Iñiguez Carchi, 2017) .

Con gran relevancia es la creación del mapeo del proceso en esta etapa, donde se documenta toda la información para entender los problemas, identificar las actividades claves que tienen mayor impacto sobre la calidad y diseñar un equipo efectivo. Herramientas útiles en esta etapa son los diagramas de Pareto, diagrama de flujo de proceso, histograma, lluvia de ideas y árbol crítico de la calidad (Garza Ríos, González Sánchez, Rodríguez González, & Hernández Asco, 2016).

Fase 2 - Medir:

Son los parámetros y actuaciones de seguimiento con el propósito de validar la información que se toma del proceso y evaluación del producto. Incluye la capacidad del proceso, los indicadores de gestión del proyecto y la medición de la satisfacción de los clientes externos e internos (Iñiguez Carchi, 2017) (Sokovic, Pavletic, & Kern Pipan, 2010) .

Herramientas como diagrama entrada-proceso-salida, análisis de capacidad de proceso, gráfico de Pareto y gráficos de control son indispensables en esta etapa, permitiendo la medición e impacto del proyecto, asimismo conocer los costos reales en contra de los costos esperados; el ciclo real del proceso contra el límite esperado y de control (Garza Ríos, González Sánchez, Rodríguez González, & Hernández Asco, 2016).

Fase 3 - Analizar:

Examinar y comprender de manera separada sus partes (variables claves) identificadas en las fases anteriores para extraer causas y determinantes del proceso. En tal sentido el equipo buscará más datos y evidencias que corroboren las causas sospechosas hasta encontrar la

[5] Descripciones cortas y simples de los requerimientos del cliente, contadas desde la posición de la persona que requiere esa nueva funcionalidad, generalmente el usuario o cliente del sistema.

causa raíz. Para ayudar a procesar dichos datos el uso de diagrama de causa efecto, análisis de varianza, muestreo, matriz de relación, correlación y regresión son herramientas disponibles para ello (Sokovic, Pavletic, & Kern Pipan, 2010) (Iñiguez Carchi, 2017) (Garza Ríos, González Sánchez, Rodríguez González, & Hernández Asco, 2016).

Fase 4 - Mejora:

Etapa de cambio del proceso y optimización del mismo al eliminar los desperdicios, al intervenir las causas que afectan al proceso, generando posibles soluciones con los datos identificados y analizados en las fases anteriores. Instrumentos como técnicas analíticas y pruebas piloto dan valor a esta fase (Sokovic, Pavletic, & Kern Pipan, 2010) (Iñiguez Carchi, 2017) (Garza Ríos, González Sánchez, Rodríguez González, & Hernández Asco, 2016) .

Fase 5 - Control:

Fase de mantenimiento y documentación de las mejoras. De igual manera es la fase de establecer los controles de verificación efectivos para los cambios del proceso que garantice la mejora alcanzada del nivel deseado mediante el uso de planes, gráficos de control o indicadores. Es indispensable definir métricas e indicadores que muestren el nivel de desempeño, de control de los objetivos y metas propuestas (Sokovic, Pavletic, & Kern Pipan, 2010) (Iñiguez Carchi, 2017) (Garza Ríos, González Sánchez, Rodríguez González, & Hernández Asco, 2016).

4.1.3 Estrategia De Testing (Pruebas)

La estrategia de testing define un conjunto de tareas, actividades, procedimientos y métodos ordenados para validar el funcionamiento de un software, buscando convencer al equipo desarrollador y al cliente que el producto es lo suficientemente bueno para su uso operacional. Esta estrategia identifica, analiza e integra técnicas, que deben volverse un hábito para el equipo, para verificar la calidad de cada uno de los componentes del desarrollo y así realizar cambios pertinentes y oportunos del mismo (Puello, 2013) (Bueno,

Crespo, & Jino, 2006) (Rojas Montes, Pino Correa, & Maur, 2015). Se constituye en uno de los aspectos más importantes del desarrollo, el cual permite validar si un producto fue construido de manera correcta, detectar puntos críticos por corregir y proporcionar información sobre la calidad del mismo (Leon Martinez, Aguilar Garcia, Vega Morales, & Gomez Florez, 2013).

Por esta razón, muchos autores concuerdan con la idea de que la estrategia de testing va más allá de la revisión final del producto desarrollado, debe ser algo dinámico y constante durante todo el ciclo de la elaboración del software, desde la revisión de los requisitos, el análisis documental, la identificación de defectos, identificación y seguimiento de riesgos hasta los planes de mejoramiento continuo (Mera Paz J. A., 2016).

A través del tiempo el concepto de testing ha ido encontrando puntos de intercepción por diferentes autores de la literatura especializada (Tabla 2), cambiando no solo su enfoque sino también su alcance hasta llegar al concepto actual que hace énfasis en verificar de forma controlada el nivel de calidad de un producto de software, prediciendo los errores y gestionando los riesgos bajo condiciones específicas.

Tabla 2. Conceptos sobre testing

AUTOR	AÑO	CONCEPTO
Dustra (Dustra, 1970)	1970	“La prueba de software puede ser usada para mostrar la presencia de bugs, pero nunca su ausencia”
Hetzel (Hetzel, 1984)	1988	"Actividad dirigida a evaluar un atributo o capacidad de un programa o sistema y determinar que cumple con los resultados requeridos"
IEEE (IEEE, 2004)	2004	“Consiste en verificar el comportamiento de un programa dinámicamente a través de un grupo finito de casos de pruebas con relación al comportamiento esperado”

E.Lewis (Lewis, 2004)	2004	“Las pruebas de software son una estrategia popular de gestión de riesgos. Se usa para verificar que se cumplieron los requisitos funcionales”
Myers (Myers, 2004)	2004	"Las pruebas de software son un proceso, o una serie de procesos, diseñados para garantizar que el código de la computadora hace lo que fue diseñado para hacer y que no hace nada sin querer. El software debe ser predecible y consistente, sin ofrecer sorpresas a los usuarios".
Bueno (Bueno, Crespo, & Jino, 2006)	2006	“La prueba de software es el proceso de ejecutar un programa de forma controlada con el objetivo de que compruebe si el programa se comporta como se define en su especificación. Es una actividad esencial para lograr un buen nivel de calidad en productos de software”.
Cárdenas y Rodríguez (Cardenas Ospina & Rodriguez Beltran, 2011)	2011	“Ayuda a investigar los errores de software, verificar la funcionalidad de los sistemas y asegurarse de que el software que se desarrolla es seguro y confiable”.

Fuente: Elaboración propia.

Las pruebas requieren una combinación de métodos alternos que sirvan como complemento para garantizar la calidad del producto, como lo es la inspección, las revisiones informales y el análisis estático, los cuales se aplican durante todo el ciclo del proceso de construcción del software, tanto a la documentación como al código fuente, con el objetivo de encontrar inconsistencias, sesgos y errores sistemáticos, de esta forma alcanzar a realizar las correcciones pertinentes en etapas tempranas y antes de que el producto sea entregado al cliente (Esterkin & Pons, 2012).

El aseguramiento de la calidad para un producto de software, no solo requiere de una buena selección de un banco de pruebas, sino de un medio planificado y sistemático para asegurar que se aplican los estándares, prácticas, procedimientos y métodos definidos en el proceso; con CMMI-DEV y la metodología DMAIC se obtienen los lineamientos para alcanzar este objetivo.

Por último, es fundamental considerar que no se puede ser probado completamente un software; Myers en los años 70s mostró un programa que contenía un loop y pocos if, resultado cien trillones de caminos, que serían probados si un tester dedicada un billón de años (Perez Lamancha, 2006) lo que se convierte en un proceso exhaustivo y poco eficiente para el desarrollo de un producto de calidad.

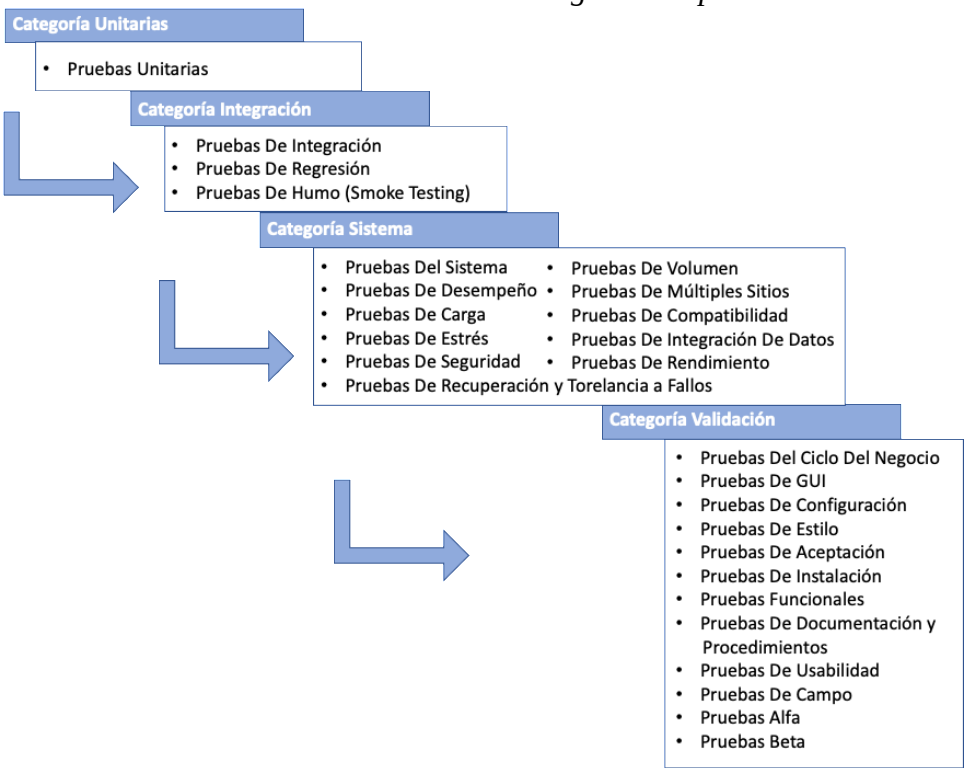
4.1.4 Tipos Y Herramientas De Pruebas

Las pruebas hacen uso de una serie de técnicas para evaluar la calidad de un producto; estas se pueden realizar sobre las funciones internas (código y estructura) de los módulos, a las cuales se le denominan pruebas de caja blanca (para cuya elaboración se debe tener amplio conocimiento de la tecnología y arquitectura utilizada), o sobre la respuesta o funcionamiento del producto (salidas) a partir de una serie de entradas, a las cuales se les denomina pruebas de caja negra (estas son realizadas generalmente desde la interfaz gráfica) (Puello, 2013) (Srinivasan, Muthu, Prasad, & Satheesh, 2014) (Software Engineering Institute, 2010).

Estas técnicas se combinan en distintas estrategias para la ejecución de las pruebas, donde, a la hora de evaluar un producto de software, se debe iniciar por los componentes más simples o pequeños e ir avanzando hasta el conjunto final o módulos complejos (Puello, 2013) (Srinivasan, Muthu, Prasad, & Satheesh, 2014) (Software Engineering Institute, 2010). La Ilustración 2**¡Error! No se encuentra el origen de la referencia.** agrupa por categorías una secuencia lógica de pruebas con la que coinciden diferentes autores para la ejecución de las mismas (Leaños Rodriguez, 2018). Esta secuencia inicia con la categoría

donde se prueba cada uno de los módulos de manera independiente (unitarias), seguidamente las pruebas de combinación de los módulos a partir del esquema del diseño (categoría integración) posteriormente el ensamble del sistema total y la comprobación de requisitos (categoría sistema) y por último la categoría validación donde el cliente comprueba que el software funciona según sus expectativas (Salazar Martínez, 2014) .

Ilustración 2. Secuencia de categorías de pruebas



Fuente: Elaboración propia.

A continuación se desarrollan las pruebas que posiblemente puedan generar mayor valor a la estrategia de testing propuesta, agrupadas por categoría.

Categoría Unitarias

Pruebas unitarias

Es la primera prueba dinámica y su diseño va dirigido a la ejecución de cada módulo de manera independiente, entendiéndola como la unidad funcional del software. En general la persona creadora del código hará esta prueba en función del contexto, localizando defectos

y comprobando el funcionamiento de los módulos del software, programas, objetos, clases, etc. (Mera Paz J. A., 2016).

Categoría Integración

Pruebas de Integración

Busca comprobar la relación de las interfaces entre los componentes o módulos del producto. Así mismo, evaluar las interacciones entre la base de datos y los subsistemas, la infraestructura, las interfaces, la configuración del sistema y los datos de configuración (Mera Paz J. A., 2016).

Categoría Sistema

Pruebas de carga

Esta prueba se encarga de verificar las tasas de transacciones, medir la capacidad y tiempos en el que el sistema da respuesta y continúa funcionando de manera correcta cuando es sometido a condiciones extremas (Jimenez Bibian, 2017) (Estayo, y otros, 2012).

Pruebas de estrés

Es tipo de prueba va dirigido a conocer la respuesta del sistema bajo condiciones anormales (mala escalabilidad, agotamiento de capacidad y memoria, fugas de memoria, condiciones de carrera, números grandes de clientes conectados en simultánea, múltiples usuarios desempeñando la misma actividad con los mismos datos) y determinar el punto ruptura del servicio (Mera Paz J. A., 2016).

Pruebas de Seguridad

El objetivo es fortalecer los niveles de acceso y verificar que un actor solo pueda acceder a las funciones y datos que su rol tiene permitido. Así mismo, las pruebas de seguridad intentan verificar que los mecanismos de protección que se construyen en el sistema en realidad lo protegerán de cualquier penetración inapropiada. Los productos de software cuyo funcionamiento gestionan datos o información sensible requieren de una protección mayor, debido a que pueden ser sujetos de ataques que vulneren su funcionamiento con el

ánimo de extraer dicha información, este motivo es el causante del uso de este tipo de pruebas que evalúen el riesgo y evitan la extracción de los datos (Mera Paz J. A., 2016) (Jimenez Bibian, 2017) .

Pruebas de Rendimiento

Estas pruebas simulan el acceso e interacción de varios usuarios, monitorea el tiempo del flujo de ejecución, acceso a datos, llamada a funciones con el propósito de determinar la velocidad de respuesta del producto e identificar los cuellos de botellas y los procesos ineficientes (Mera Paz J. A., 2016) (Jimenez Bibian, 2017).

Categoría Validación

Pruebas de aceptación

Consisten en validar los criterios expuestos en el contrato de desarrollo del producto por parte del cliente, los requisitos dispuestos por el cliente contra las funcionalidades presentes en el producto (Mera Paz J. A., 2016). En este tipo de pruebas el cliente realiza tareas típicas para comprobar que se satisfacen los requisitos y así aceptar o no el producto de software. Estas deben ir acompañadas del manual del usuario y del administrador (Jimenez Bibian, 2017) (Mosquera Diaz, Ocampo Cortezz, & Garcia Monsalve, 2014) (Flebes Estrada, y otros, 2011).

Pruebas Funcionales

Estas pruebas incluyen la navegación, entrada de datos, procesamiento y obtención de resultados, tiene como meta verificar la aceptación de datos, el procesamiento, recuperación y la implementación adecuada de las reglas del negocio.

El aseguramiento de la calidad del producto de software no solo requiere de una buena selección de un banco de pruebas sino de un medio planificado y sistemático para asegurar que se aplican los estándares, prácticas, procedimientos y métodos definidos del proceso conocido como herramientas de pruebas que al lado del modelo CMMI-DEV y la metodología DMAIC aseguren la calidad del producto de software.

4.2 MARCO CONTEXTUAL

La empresa Ceiba software es una organización colombiana de carácter privada que tiene como misión trabajar con pasión para transformar los negocios de sus clientes, haciéndolos más exitosos mediante el uso de la tecnología y la innovación.

Consolidada hace 15 años, cuenta con un talento humano de aproximadamente 400 personas y fue categorizada por el centro de investigación laboral CINCEL como una empresa con los mejores climas organizacionales, desde la concepción de las ideas y el diseño de la experiencia hasta la materialización vía tecnológica.

Es una empresa pionera de la agilidad en Colombia desde hace 9 años y los únicos avalados por Scrum Alliance, certificada en nivel 5 de CMMI-DEV. Actualmente exporta soluciones a clientes en USA, España, Argentina, Canadá.

Los proyectos de Desarrollo de Software en la compañía son ejecutados bajo el marco de trabajo Scrum, donde se realiza una recolección de necesidades prioritarias para el cliente, estas son detalladas y llevadas a una lista de trabajo pendiente mediante historias de usuario, las cuales luego son divididas y priorizadas en Sprints[6], con una duración máxima de dos semanas para la mayoría de los proyectos, generando al final de cada Sprint entregables de valor para el cliente.

La estrategia de testing (pruebas) empleada en la empresa está implícita en el proceso de desarrollo de software, la cual es ejecutada por un desarrollador, siendo un requisito más para el equipo, donde se deben aplicar y ejecutar herramientas de prueba para el funcionamiento del producto. Todo este proceso se encuentra incluido dentro de metodologías ágiles, donde el desarrollo es dinámico y dirigido a respuestas rápidas y flexibles de autogestión para la solución de los problemas.

6 Iteración de tiempo dentro de un proyecto, cuyo objetivo es la entrega de un producto incremental que genere valor.

Dentro del proceso de desarrollo definido en la compañía, se encuentra la etapa de la ejecución del sprint, en ella se contemplan las fases de diseño, construcción e implantación del software. La etapa de construcción, que para efectos del proyecto es la que se tomará como referencia, tiene como objetivo desarrollar partes del software de manera eficiente en base a lo especificado por el usuario; el equipo de desarrollo debe generar los componentes del código teniendo en cuenta las buenas prácticas de desarrollo de software, las cuales incluyen código legible, limpio y documentado, y teniendo en cuenta los atributos de calidad dentro de los cuales se indica que se debe satisfacer las necesidades del cliente, que la funcionalidad esté disponible para su uso en cualquier momento, que sea intuitiva y fácil de usar, segura y escalable.

Para garantizar lo mencionado anteriormente, cada proyecto que se ejecuta en la compañía debe contar con un plan de calidad que a su vez define un plan de pruebas para la estrategia de testing, el cual tiene por objetivo:

Tomado del plan de calidad del sitio del proyecto de Ceiba Software:

- Cumplir con los criterios de aceptación[7] de las historias de usuario definidas.
- Cumplir con criterios de calidad definidos para cada historia de usuario.
- Generar satisfacción en el cliente y en el equipo con la calidad del producto entregado.
- Garantizar la mantenibilidad del código según la arquitectura planteada.

En la estrategia de testing se incluyen las técnicas y tipos de pruebas que se van a realizar (Tabla 3).

7 Son los requisitos del cliente sobre cómo debe ser el funcionamiento del componente para que una determinada acción se pueda realizar.

Tabla 3. Tipos de pruebas de Ceiba software

TIPO DE PRUEBAS	TÉCNICAS Y/O HERRAMIENTAS
Pruebas unitarias	MSTest, Nsubstitute para Backend, Karma, Jasmine, *Unit Test.*
Pruebas funcionales (integración)	MSTest
Pruebas de Rendimiento	SAP/QUO
Carga	Jmeter
Pruebas de Seguridad	OWASP
Revisión de código (estática)	Sonar y Team Foundation System
Servidor de integración continua	Jenkins y Team foundation system

Fuente: Plan de calidad del sitio del proyecto de Ceiba Software

Dentro del plan de calidad también se considera el plan para la revisión par (inspección) en la cual se evalúan los siguientes temas: estética del código, arquitectura, estado de la aplicación, manejo de errores, seguridad, pruebas funcionales, pruebas de integración, banckend y frontend en general, pruebas unitarias y de rendimiento, los resultados se consolidan y se registran en un documento propio de la compañía llamado *"informe para la revisión par"*.

Los items mencionados anteriormente son revisados por un par experto de la compañía quien generalmente es externo al proyecto, el cual se encarga de validar que se esté teniendo en cuenta cada tema al momento de generar un producto de calidad.

5 OBJETIVOS

5.1 OBJETIVO GENERAL

Mejorar la estrategia de testing involucrada en el proceso de desarrollo de la empresa Ceiba Software, mediante la metodología DMAIC, alineado con las áreas PPQA, VAL y VER de CMMI-DEV para la disminución de desperdicios[8] y el mejoramiento de la calidad de los productos.

5.2 OBJETIVOS ESPECÍFICOS

- Diagnosticar el estado actual de la estrategia de testing en la empresa Ceiba Software, siguiendo los lineamientos de la metodología DMAIC, para identificar los desperdicios que están generando mayor impacto.
- Diseñar e implementar un modelo integrador entre la metodología DMAIC y las áreas PPQA, VER y VAL de CMMI-DEV, disminuyendo los desperdicios encontrados en el objetivo anterior.
- Medir y analizar los resultados encontrados en el diagnóstico inicial y de la propuesta con el fin de determinar la validez del modelo propuesto.
- Proponer indicadores de seguimiento que generen valor a la estrategia de testing.

[8] Cualquier cosa involucrada en la ejecución de un proceso o desarrollo de un producto que no genere valor final al cliente interno y externo.

6 METODOLOGÍA

6.1 ENFOQUE Y MEDOLOGÍA

Para el desarrollo del proyecto se utilizó una investigación aplicada, la cual consiste en poner en práctica la teoría y los conocimientos previamente obtenidos, con el propósito de mejorar la estrategia de testing en la empresa Ceiba Software.

El proyecto se realizó en 5 fases, las cuales son equivalentes con las fases de DMAIC (Ilustración 3), la primera (Definir) donde se realizó el diagnóstico del estado actual del proceso de testing en la empresa Ceiba Software, en esta se identificó la información clave que sirvió para realizar los análisis respectivos.

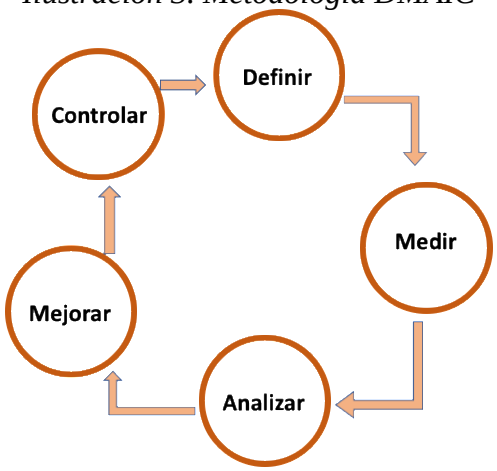
En la segunda y tercera fase (Medir – Analiza), se hizo el tratamiento de los datos obtenidos y se identificaron los desperdicios de la estrategia actual para convertirlos en acciones de mejoramiento a utilizar en la fase siguiente.

En la cuarta fase (Mejorar) se diseñó un modelo enmarcado en el ciclo de la metodología DMAIC, este como producto de realizar una búsqueda en bases de datos científicas(Google academic, Scielo, Ebsco, Redlyc, Science Direct) usando como palabras claves CMMI, VER, VAL, PPQA, DMAIC, Estrategia de testing, Pruebas de software, Mejoramiento de procesos y Mejora continua, en español e inglés, con el propósito de extraer las buenas prácticas utilizadas en diferentes industrias y en especial en la ingeniería de software para el mejoramiento de una estrategia y de esta forma construir un modelo sistemático de mejoramiento continuo. En él se incluyeron las 3 áreas de CMMI que tienen relación directa con la estrategia de testing incorporando una serie de pruebas que han tenido buen resultado según lo encontrado en el marco teórico y adaptándola según las buenas prácticas encontradas en las dos fases anteriores.

El modelo se aplicó como piloto en un proyecto de la empresa, para corroborar su efectividad se comparó la estrategia actual de la organización con el modelo propuesto. En

la fase quinta (Controlar) se propuso una serie de métricas las cuales fueron adaptadas de diferentes áreas del conocimiento encontradas en la revisión bibliográfica que han aportado y dan cuenta del mejoramiento de procesos a través del tiempo, estas fueron aplicadas en el proyecto piloto y validadas por un líder de procesos.

Ilustración 3. Metodología DMAIC



Fuente: Elaboración propia

6.2 PROCEDIMIENTO DE RECOLECCIÓN DE INFORMACIÓN

Se solicitó la autorización a la empresa Ceiba Software S.A. para el desarrollo del proyecto investigativo y acceso a la información secundaria (Proceso y Procedimiento).

Se aplicó un cuestionario tipo encuesta (Anexo 1) la cual contenía 12 preguntas sobre la temática desarrollada en este proyecto en una muestra representativa de desarrolladores mediante un formulario de Google de manera individual y garantizando la confidencialidad de la información; se realizó una revisión de información secundaria del repositorio del sitio del proyecto para conocer la metodología y el proceso documentado utilizado en la ejecución del mismos. Se realizó la verificación de la estrategia actual en un proyecto mediante el uso de una lista de chequeo con 17 criterios previamente definida (Anexo 2).

Se solicitó acceso al repositorio donde se encuentra la información de los proyectos en curso y entregados, de allí se tomó un proyecto cuya ejecución estuviera entre las 3 y 6

semanas con el fin de utilizar el modelo propuesto, al que se le aplicó nuevamente la lista de verificación (Anexo 2) para comparar los resultados encontrados en ambas listas que al ser proyectos similares permitió validar la efectividad del mismo.

6.3 ORDENAMIENTO DE LA INFORMACIÓN Y PLAN DE ANÁLISIS

La información recolectada de la encuesta y la lista de chequeo se consolidó en una base del programa Microsoft Excel.

A las preguntas ¿Qué tipo de pruebas considera que son indispensable para un proceso de testing? y ¿Qué tipo de pruebas realiza actualmente? Se agruparon los tipos de pruebas según su categoría (Ilustración 2**¡Error! No se encuentra el origen de la referencia.**) para su análisis y junto con la pregunta ¿cuál es el porcentaje de cobertura definido para su proyecto? se les realizó un análisis univariado mediante el uso de la estadística descriptiva.

Con las respuestas a la pregunta ¿cuáles son los bloqueantes o problemas en el proceso de testing?, con el ánimo de conocer los desperdicios de la estrategia actual que afectan al cliente interno y externo, se clasificaron en 10 problemas y se construyó una Matriz de Vester para su análisis. Este método permitió determinar la correlación de los problemas relevantes y determinar problemas críticos, pasivos e irrelevantes que afectan la estrategia de testing. Se construyó la matriz mediante una triangulación de los problemas que aparecen en las filas versus los problemas que aparecen en las columnas. Basada en la pregunta: ¿Cómo influye el problema 1 sobre el problema 2? y calificando su grado de influencia (Tabla 4) sucesivamente hasta completar la matriz de dependencia e influencia (Anexo 3).

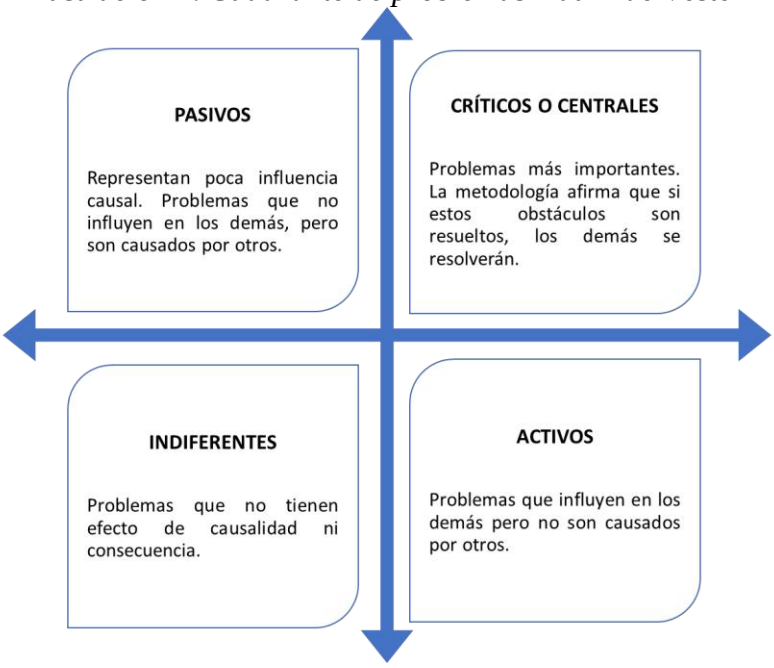
Tabla 4. Grado de influencia

Valor	Descripción
0	No existe relación directa
1	Existe una influencia débil
2	Existe una influencia mediana
3	Existe una influencia fuerte

Fuente: Elaboración propia

En resumen, se determinan los valores de X y Y para cada problema y de esta manera se construyó el plano cartesiano, ubicando cada uno de los problemas en su cuadrante correspondiente e interpretado según la Ilustración 4.

Ilustración 4. Cuadrante de problemas matriz de Vester



Fuente: Elaboración propia

Posteriormente se aplicó la lista de verificación (Anexo 2) en los dos proyectos, el primero con la estrategia actual de la organización y el segundo con el modelo propuesto, a los cuales se les realizó un análisis cualitativo comparando ambos proyectos y las buenas prácticas de las áreas de CMMI-DEV tenidas en cuenta para el proyecto.

7 RESULTADOS

El presente capítulo expone y analiza los resultados del trabajo investigativo y aplicado, el cual se divide en 5 secciones acorde con la metodología DMAIC seguida (7.1 Definición de la estrategia actual; 7.2 Medición y análisis inicial; 7.3 Diseño e implementación de la mejora; 7.4 Medición y análisis secundario; 7.5 Control y seguimiento).

7.1 DEFINICIÓN DE LA ESTRATEGIA ACTUAL

El diagnóstico de la estrategia de testing en la empresa Ceiba Software surge a partir de un análisis interpretativo de la información recolectada en la encuesta (Anexo 1) y revisión de documentación secundaria.

Con el fin de conocer la percepción por parte de los desarrolladores sobre las pruebas que consideran esenciales dentro de una estrategia de testing en contraste con las que actualmente realizan, se construyó la Tabla 5 la cual presenta las pruebas agrupadas según las categorías tratadas en este estudio (Ilustración 2). Cada una de las preguntas contiene una columna "categoría" la cual indica el número de encuestados que seleccionaron al menos un tipo de prueba de la misma y la columna "tipo de prueba" la cual representa la proporción de personas que seleccionaron cada tipo de prueba de manera porcentual y gráfica.

La Tabla 5 muestra que las categorías unitarias, integración y sistema son indispensables para el 100% de la muestra, pero el 87% realiza pruebas de la categoría unitarias, el 63% de la categoría integración y solo el 45% de la categoría sistema. El 95% de la muestra considera importante la categoría validación, pero 6 de cada 10 las realiza (61%).

Ya que el 100% de la muestra considera indispensable la ejecución de las pruebas unitarias se demuestra la pertinencia de la pregunta realizada en la encuesta ¿Sabe usted cuando una prueba unitaria no genera valor?, a lo que el encuestado E17 respondió "Si no está sirviendo para validar el comportamiento esperado por el componente, la prueba debe

replantearse. Lo importante es no solo validar la ruta feliz (o resultado esperado) sino también la ruta o rutas de excepción.”, el encuestado E14 indico que “cuando no se está cubriendo una lógica de negocio o funcionalidad complementaria para llevar a cabo un cálculo o proceso requerido para el negocio” el encuestado E21 respondió que “Cuando solo se hacen por satisfacer cobertura”. Estas consideraciones se traducen en que una prueba unitaria no genera valor cuando no se evalúa el contexto del negocio y no permite detectar errores en los flujos alternos y su ejecución solo se hace con el objetivo de aumentar la cobertura.

Tabla 5. Consolidado de los tipos de pruebas indispensables por categoría (%)

Categoría	Tipo De Pruebas	Pruebas indispensables		Pruebas que se realizan	
		Categoría	Tipo de prueba	Categoría	Tipo de prueba
Unitarias	Pruebas unitarias	100%	100%	87%	87%
Integración	Pruebas de integración	100%	97%	63%	66%
	Pruebas de regresión		37%		11%
Sistema	Pruebas cruzadas	100%	0%	45%	3%
	Pruebas de carga		76%		26%
	Pruebas de compatibilidad		16%		5%
	Pruebas de desempeño		21%		3%
	Pruebas de estrés		55%		8%
	Pruebas de integración de datos		29%		3%
	Pruebas de múltiples sitios		3%		0%
	Pruebas de performance		0%		3%
	Pruebas de rendimiento		71%		8%
	Pruebas de seguridad		74%		24%
	Pruebas de volumen		8%		3%
	Pruebas del sistema		16%		3%
	Pruebas de recuperación y tolerancia a fallos		18%		3%
Validación	Pruebas funcionales	95%	84%	61%	53%
	Pruebas de aceptación		50%		37%
	Pruebas de configuración		8%		0%
	Pruebas de estilo		3%		0%
	Pruebas de GUI		26%		3%
	Pruebas de usabilidad		24%		0%

Fuente Elaboración Propia

La categoría integración está compuesta por dos tipos de pruebas, integración propiamente dicha y regresión, donde el 97% de la muestra considera indispensable la primera y solo el 37% la segunda, pero cabe resaltar que las pruebas unitarias y de integración son las que más realizan (87% y 66% respectivamente) en la organización.

La categoría sistema presenta un gran número de tipos de pruebas, donde las pruebas de seguridad, carga y rendimiento presentaron los porcentajes más altos dentro de la misma

(76%, 74%, 71%), cabe resaltar que las pruebas de carga y seguridad son las que más llevan a cabo los desarrolladores con un 26%, 24% respectivamente. Se evidencia que las pruebas cruzadas y de performance no fueron consideradas como pruebas relevantes dentro de una estrategia de testing para la muestra, pero a pesar de esto el 3% las realiza.

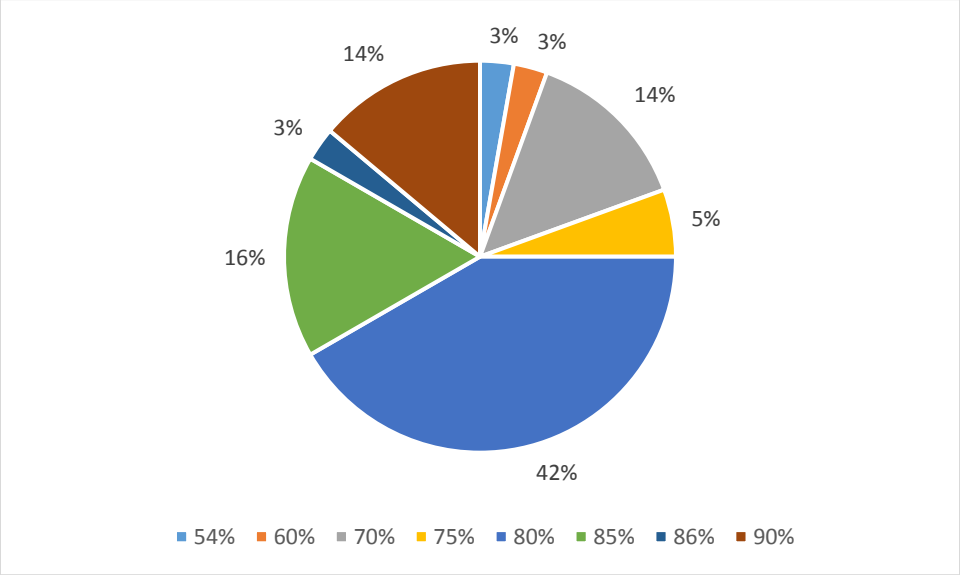
Dentro de la categoría validación 6 tipos de pruebas fueron consideradas indispensables para una estrategia de testing, con un mayor porcentaje para las pruebas funcionales (84%) y de aceptación (50%), donde a su vez presentaron porcentajes altos en las personas que actualmente las realizan (53% y 37%), pero en este solo se realizan 3 tipos de pruebas. Este análisis mostró que del total de la muestra el 39% no consideran indispensable realizar algún tipo de pruebas de esta categoría.

El 95% de la muestra considera que se debe realizar mínimo un tipo de prueba por categoría, el 23,6% considera que un buen proceso de testing debe incluir 11 o más tipos de pruebas, aproximadamente 7 de cada 10 (73,6%) considera que se deben hacer más de 5 y hasta 10 tipos de pruebas. Al indagar él porque del conjunto de pruebas seleccionadas se encontró que el encuestado E19 considera "Son los atributos que permiten tener un sistema robusto escalable", E23 dice "Las pruebas a aplicar dependerán del nivel de riesgos, la arquitectura y tecnologías que se involucren en la solución. Es por esto por lo que se deben definir muy bien que pruebas se deben realizar para entregar un producto con buenos estándares de calidad.", E15 "Porque permiten crear un sistema seguro, robusto y con buen desempeño en situaciones reales de uso." y el encuestado E26 "Se cubre la funcionalidad correcta del desarrollo, velocidad de respuesta, se comprueban flujos completos del negocio, nivel de seguridad, comunicación de los distintos componentes dentro de la aplicación." Lo anterior indica que para un buen desarrollo de un producto de calidad es necesario realizar de cada categoría mínimo un tipo de prueba y entre más tipos de prueba se tengan se logra asegurar la funcionalidad correcta del mismo.

Frente a la cobertura entendida como el grado de profundidad que tiene testeada una funcionalidad, se encontró que el 5% no tiene definido un porcentaje, el restante lo tiene

distribuido de la siguiente manera, el 43% de la muestra contempla un porcentaje de cobertura del 80%, el 16% un porcentaje del 85% y el 14% un porcentaje del 90%, como lo muestra la Ilustración 5.

Ilustración 5. Porcentaje de cobertura



Fuente Elaboración Propia

De lo anterior se puede concluir que el promedio definido por la muestra para la cobertura es de 79,4% con una moda del 80% y un máximo de 90%.

Llama la atención que el porcentaje más bajo de cobertura encontrado fue 54%, donde al revisar la respuesta a la pregunta ¿Sabe usted para que sirve la cobertura y porque se definió ese valor para el proyecto? El entrevistado E4 respondió "El proyecto empezó sin pruebas unitarias", lo que muestra que un proyecto que inicio sin pruebas cuando ya se encuentra tan avanzado tiene un grado de complejidad mayor llevar la cobertura a más del 79,4% como lo indica el promedio.

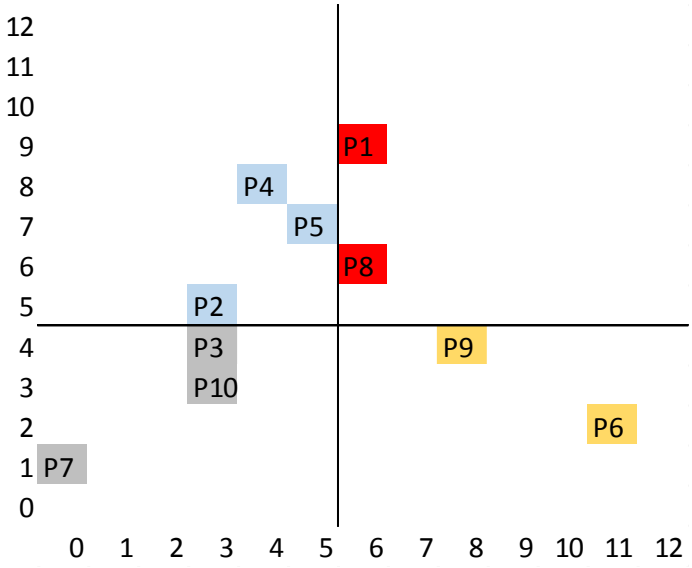
Se indagó sobre las pruebas que ejecuta actualmente cada desarrollador de la muestra y cuales no le generan valor, encontrando que el 100% coincidió en que las que realizan dan valor, por ejemplo el encuestado E18 indica que "Todas las pruebas que se hacen generan valor al negocio, ya que de una u otra forma están probando un aspecto de la aplicación,

desde su rendimiento, hasta su usabilidad, pasando por seguridad", el E25 indica que "Cada tipo de prueba se realiza con un objetivo diferente por lo que todas generan algún valor." a diferencia de lo anterior, el encuestado E38 menciona "Algunas de las pruebas unitarias realizadas no generan valor porque solo sirven para aumentar cobertura".

Referente a los problemas, bloqueantes, impedimentos frecuentes con los que se enfrentan los desarrolladores en la empresa objeto del estudio, considerados como desperdicios de la estrategia, se agruparon en 10 problemas comunes (P1, P2, P3…P10), se realizó una matriz de dependencia e influencia (Anexo 3) y posteriormente se grafica un plano cartesiano de Vester. Con el ánimo de conocer el papel que juega la estrategia de testing dentro del proceso de desarrollo de software, se parte de conocer los principales errores, problemas o dificultades que se presenta de manera más común en la empresa, su identificación y conocimiento de los mismos fue el insumo primordial para adaptar el modelo de mejora para la estrategia de testing.

La Ilustración 6 representa de forma gráfica como se distribuyen los problemas según la herramienta de matriz de Vester, donde se encontraron 2 problemas en el cuadrante crítico, 2 en el activo, 3 en el pasivo y 3 en el indiferente.

Ilustración 6. Plano cartesiano de Vester



Fuente: Elaboración propia

Los problemas centrales encontrados fueron, un alto acoplamiento en el sistema -refiriéndose al grado de dependencia que existe entre los diferentes módulos de un producto-, y el desconocimiento del negocio -cuando no se tiene la información suficiente de los temas a los que se le debe dar solución. Sobre estos dos problemas el encuestado E16 indica que "Cuando se hacen después de diseñar las clases y hay muchas dependencias en las clases, se vuelve engorroso", el E3 "Un mal diseño del sistema, donde se acopla con librerías que no permiten realizar pruebas", el E19 "Problemas conceptuales son los más evidentes, pero también desconocimiento".

Dentro de los problemas encontrados en el cuadrante activos está la falta de tiempo para realizar las pruebas, esto ocurre cuando dentro de la estimación inicial no se contempla el esfuerzo que el equipo de desarrollo va a invertir testeando el producto final y el exceso de pruebas, entendido como la falta de claridad en la planeación de las mismas, lo que genera, reprocesos o pruebas que no generan valor, referente a esto se encontró que el encuestado E14 indica "disponibilidad de tiempo porque no se contempla en las historia de usuario ese tiempo", el E9 "falta de tiempo".

Tabla 6. Resultados de problemas por cuadrante

Cuadrante	ID del problema	Nombre del problema
Críticos Centrales	P1, P8	Un alto acoplamiento en el sistema. Desconocimiento del negocio.
Activos	P6, P9	Falta de tiempo para realizar las pruebas. Exceso de pruebas.
Pasivos	P2, P4, P5	Métodos anidados difíciles de "mockear". Falta de documentación del sistema. Falta de definiciones funcionales.
Indiferentes	P3, P7, P10	Falta de conocimiento en pruebas de software. No contar con un ambiente adecuado para las pruebas.

		No contemplación de tiempos de pruebas en la propuesta comercial.

Fuente: Elaboración propia

La Tabla 6 muestra los problemas pasivos, donde la falta de documentación del sistema no permite conocer los proceso y procedimientos involucrados en el producto, además de ello se le dificulta al equipo conocer que función cumple el código con el que está desarrollado y aun más cuando los métodos están estrechamente asociados entre sí y son difíciles de "mockear" y en los problemas indiferentes se encontró la falta de conocimiento sobre los diferentes tipos de pruebas existentes, no contar con un ambiente adecuado de prueba y no contemplar los tiempos de pruebas en la propuesta comercial, lo que traduce que aunque estos problemas no son causa ni efecto directo de los demás están latentes dentro de los impedimentos frecuentes.

Se indagó sobre si se debería mejorar el área actual de testing y si los encuestados consideran que el hecho de contar con talento humano especializado en pruebas mejoraría la calidad del software desarrollado teniendo en cuenta que el desarrollador es la persona que ejecuta las pruebas, a lo que se encontró que el 65% de la muestra coincide con la respuesta "sí" en ambas preguntas, como se observa en la Tabla 7, el 86% considera que se debe contar con personal especializado en pruebas y el 73% de la muestra que se debe realizar mejoramiento a la estrategia actual de testing.

Tabla 7. Asociación entre el mejoramiento de la estrategia testing y el talento humano especializado.

		Área con talento humano especializado en pruebas		**TOTAL**
		SÍ	NO	
Mejoramiento del proceso de testing	SÍ	23	5	**28**
	NO	10	0	**10**
TOTAL		**33**	**5**	**38**

Fuente: Elaboración propia

Con fines académicos se realizó una prueba estadística de *chi cuadrado* para determinar si existe una asociación entre considerar un mejoramiento del proceso de testing mediante la creación de un área especializada, a lo que no se encontró valores estadísticamente significativos, con un valor de *chi cuadrado* de la tabla observada de 2,05 siendo un valor menor al de referencia de la tabla de 3,84.

7.2 MEDICIÓN Y ANÁLISIS INICIAL

En este apartado se encuentran los resultados y análisis de la aplicación de la lista de chequeo para la evaluación de la estrategia de testing (Anexo 2) la cual fue aplicada a un proyecto en su fase terminal cuya duración total fue de 4 meses, equivalentes a 6 sprints, este proyecto tuvo como objetivo realizar una aplicación web responsive que permitiera que un cliente de tipo persona Natural pudiera tener la posibilidad de firmar el contrato y pagaré de manera electrónica, esto se logró a través de la integración con Certicámara para validación de identidad, firma de contrato, Deceval para la generación y firma de pagaré desmaterializado.

Como alcance para el proyecto se determinó disminuir los tiempos para firmar contratos asociados al negocio de persona natural, incrementar la seguridad en la custodia de documentos legales y agilizar la consulta de estos, además de disminuir la operatividad en los procesos internos.

Como resultado global después de aplicar la lista de chequeo se obtuvo un 70% de criterios cumplidos, resaltando que para el proyecto se definieron claramente los objetivos de la estrategia de pruebas y se almacenó en el sitio del proyecto, dentro de los que se encontraba, cumplir con los criterios de aceptación y criterios de calidad definidos para cada historia de usuario, generar satisfacción en el cliente y en el equipo con la calidad del producto entregado, garantizar la calidad del producto entregado al cliente al finalizar cada sprint, en la implantación y la mantenibilidad del código según la arquitectura planteada.

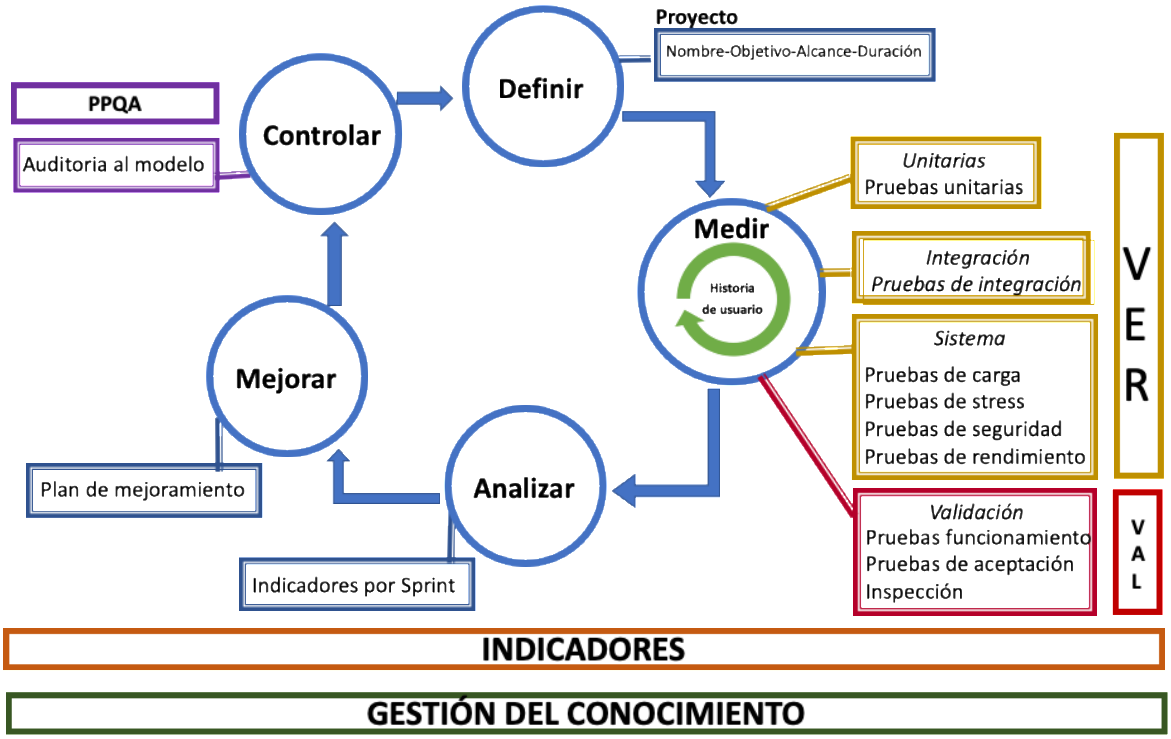
El proyecto además contaba con el siguiente objetivo para la estrategia de testing, especificar los tipos de pruebas a incluir y técnicas a utilizar en cada caso. Dentro de los criterios evaluados en el proyecto se encontró que el 36% no cumplían, por ejemplo, solo el 20% de los tipos de pruebas que se ejecutaron tenían definidas las herramientas en la cuales se iban a realizar y de los 6 tipos de pruebas definidas se realizaron 5, de las cuales no existe evidencia de la totalidad de los resultados.

No se encuentra definido un plan dentro del proyecto que cuente con toda la información necesaria al momento de llevar a cabo una capacitación que contemple todos los temas de pruebas.

7.3 DISEÑO E IMPLEMENTACIÓN DE LA MEJORA

Esta sección describe la propuesta del modelo (Ilustración 7) la cual es una representación conceptual y gráfica para el mejoramiento de la estrategia de testing en la compañía Ceiba Software. Su construcción parte del ciclo DMAIC e integra las áreas de VAL, VER y PPQA de CMMI-DEV; el modelo surge como resultado del análisis de las buenas prácticas encontradas en el marco teórico para posteriormente ser adaptado a partir de los resultados encontrados en las secciones anteriores (7.1 y 7.2) y de esa manera adoptarlo en la estrategia actual de la empresa objeto.

Ilustración 7. Modelo de testing



Fuente: Elaboración propia

Todos los resultados del modelo se deben ir consolidando en el anexo informe de la estrategia y plan de mejora (

Anexo 4,

Anexo 5) con el fin de dejar la evidencia de la estrategia en el proyecto e ir construyendo un repositorio de cada uno, eliminando de esta manera la subjetividad de la información al ser anecdótica por no tener soporte documentado.

El modelo parte de una fase de definición, en esta se reúne la información básica e inicial del proyecto a desarrollar donde se precisa el nombre, objetivo, alcance, duración del mismo, esta última se construye en base a los requerimientos del cliente, acompañado de un experto del área comercial y un desarrollador con buenos conocimientos y habilidades, con el fin de determinar una cantidad en días/meses razonables para la ejecución del proyecto. Además se describe la duración estimada para las pruebas y tipos de pruebas recomendadas las cuales son establecidas por el gerente del proyecto y el equipo de desarrollo, en la sesión de socialización del proyecto.

Teniendo la información general del proyecto se estima el total de sprints a ejecutar y la duración de los mismos; el apartado sprints reales ejecutados se registra al finalizar el proyecto, este aplica cuando los proyectos están bajo la modalidad ágil.

Todas las observaciones que surgen de esta reunión y puedan impactar en la estrategia de testing deben ser consolidadas en el apartado observaciones del anexo 4, es aquí donde se debe aclarar cuáles tipos de pruebas del modelo no se pueden utilizar debido a la naturaleza del proyecto.

Seguidamente damos paso a la fase de medir, aquí se centra la ejecución del ciclo de pruebas las cuales son realizadas en un ambiente previamente dispuesto para la ejecución de las mismas (ambiente de pruebas) para cada historia de usuario o requisito según la metodología definida por la organización. Este ciclo está diseñado de manera escalonada según la categoría, lo que indica que parte de probar y asegurar el correcto funcionamiento de un módulo de código (categorías unitarias) hasta la revisión del software en conjunto que cumpla con las especificaciones y lo que el usuario realmente quiere (categoría validación).

El área de VER está compuesta de las 3 primeras categorías de pruebas: unitarias con tipo de pruebas unitarias; integración con el tipo de prueba de integración y sistema con los tipos de pruebas de carga, estrés, seguridad y rendimiento, con el objetivo de evaluar que el producto se esté desarrollando correctamente.

Seguidamente el área de VAL está conformada por la categoría validación que incluye los tipos de pruebas funcionales y aceptación con el propósito de verificar que el producto desarrollado cumpla con los requisitos definidos por el cliente y de igual manera se incluye la estrategia de inspección, llevada a cabo por medio de una evaluación de un par (Anexo 6).

El recorrido anteriormente mencionado cumple con el objetivo de lograr disminuir el número de errores o desperdicios a medida que se avanza en el desarrollo del producto y en producción, por lo tanto, se debe asegurar la realización de las pruebas de cada categoría de manera secuencial. Cabe aclarar que se proponen categorías que se componen de varios tipos de pruebas, para este caso el orden de ejecución no altera el éxito del modelo.

Como producto de la fase medir, se obtiene la cantidad de errores encontrados, cantidad de errores solucionados, el tiempo invertido en la corrección datos para la construcción de indicadores operativos. Para cada uno de los errores o incidentes encontrados se deja el registro detallado del mismo, su clasificación en el anexo 4 apartado clasificación de errores.

Para la clasificación de los errores el desarrollador selecciona entre alto, medio, bajo según el impacto (nivel de daño que causa en la organización) y la urgencia (prioridad con la que debe ser resuelto el incidente), donde el resultado de la intercepción de las dos variables indica la criticidad, como se observa en la Tabla 8.

Tabla 8. Clasificación de los errores

Impacto	Alto	Medio	Bajo

Urgencia			
Alto	1	2	3
Medio	2	3	4
Bajo	3	4	5

Fuente ISO 27001

Los resultados de la clasificación van de 1 a 5, siendo 1 y 2 los errores más críticos que requieren mayor atención y una solución oportuna y los valores más cercanos a 5 son los de menor prioridad para la solución del error. Estas soluciones se registran en el anexo 4 sección medir apartado clasificación de errores. En este mismo apartado se establece el estado como cerrado, o si no se ha dado alguna solución se registra como abierto en la columna estado.

En la fase analizar, destinada a evaluar las métricas operativas con los datos generados en la fase anterior como proporción de tiempo de esfuerzo por Sprint, promedio de tiempo invertido en la solución de errores, índice de errores reportados en el proyecto, proporción de errores abiertos en el proyecto, proporción de errores críticos abiertos en el proyecto y efectividad de pruebas según la categoría (ver ficha técnica en el apartado 7.5), para analizarlas frente a las metas establecidas y tomar decisiones basadas en ellas y sus desviaciones. La definición de metas para los indicadores y rangos aceptables permitirá a la organización y de manera particular al equipo del proyecto, identificar aquellas actividades que están requiriendo mayor esfuerzo y por consiguiente contemplar la posibilidad de involucrar más talento humano.

La fase mejorar busca identificar las oportunidades de mejora y convertirlas en acciones que generen valor a la estrategia de testing y disminuyan el número de desperdicios, plasmándolos en un plan que permita registrar la evolución del mismo por medio de seguimientos el cual queda registrado en el

Anexo 5, indicando la fuente que generó el hallazgo (indicador fuera de la meta, inspección, auditoria al proceso) y registrando la acción a implementar, responsable y plazo los cuales serán revisados con una periodicidad de 3 meses por el líder del proceso. Por último, es necesario el control del modelo y para ello la auditoría al proceso juega un papel importante, ya que ésta permite velar por la calidad de la estrategia y del producto, de esta forma se incluye el área de PPQA de CMMI-DEV en la fase controlar la cual hace uso de la lista de verificación (Anexo 2) para dicho proceso.

Como insumo al modelo se incluye una tabla consolidada de consulta con diferentes herramientas para la ejecución de cada una de las pruebas a utilizar dependiendo del lenguaje de desarrollo en la fase medir, donde se detectó como una oportunidad de mejora para la estrategia actual de la organización ya que no contaban con ella.

Tabla 9. Herramientas por tipo de prueba

Tipo de prueba	Herramienta	Tecnología de programación
Pruebas Unitarias	Junit	Java
	Mockito	
	Reactor Test	
	Jest	JavaScript
	Mocha Chai	
	Jasmine	Angular
	Karma	
	Scalatest	Scala
	Mockito	
	Spec2	
	Enzyme	React
	MSTest	C#
	Ractor Test	Reactor
Pruebas Funcionales	Katalon	Web/Mobile/Desktop
	Serenity BDD	Java
	Jbehave	
	Cucumber	
	Cucumber	Scala
	Protactor	Protactor
	Cypress.io	React
	Selenium	Cualquier tipo de tecnología
Pruebas de Integración	Scalatestplus-play	Scala
	Wiremock	Java
Pruebas de Aceptación	Serenity BDD	Java
	Postman	
	SoapUI	

	Insomnia-Moockon	Cualquier tipo de tecnología especialmente donde se expongan servicios web
	Screenplay	Cualquier Framework de desarrollo
Pruebas de Carga	Gatling	Cualquier tipo de tecnología especialmente donde se expongan servicios web
	Jmeter	Cualquier tipo de tecnología especialmente donde se expongan servicios web
Pruebas de estres	Jest	Java
	Performance Tester	
	Jmeter	
Pruebas de rendimiento	Jmeter	Java
Librería para generar cobertura de pruebas unitarias	Jacoco	
Integración Continua	Jenkins	Java
	Jenkins	Sacala
Pruebas de Seguridad	OWASP Zap Prox	Cualquier tipo de lenguaje de desarrollo

Fuente: Elaboración propia

El cierre del modelo esta dado por los siguientes indicadores estratégicos, proporción de satisfacción global, costo de soluciones de errores del proyecto(ver ficha en el apartado 7.4) y la gestión del conocimiento, los cuales permiten el crecimiento de la cultura organizacional en los temas de testing al dejar disponible y documentado las lecciones aprendidas derivadas de la solución de los errores.

7.4 MEDIR Y ANALIZAR SECUNDARIO

Esta sección muestra los resultados de la aplicación del Anexo 2 en un proyecto en el cual se desarrolló el modelo en 2 sprints, y de esta manera comparar los 3 apartados anteriores (7.1,7.2 y 7.3) reconociendo las fortalezas y debilidades del mismo.

Cuando se aplicó la lista de chequeo a la estrategia actual (7.2) dio como resultado un 70% de cumplimiento sobre los criterios evaluados, mientras que al aplicar el mismo

instrumento en el proyecto que utilizó el modelo propuesto, presentó un cumplimiento del 94%, estos 20 puntos porcentuales por encima muestran el mejoramiento que puede tener un proyecto aplicando la estrategia propuesta en la estrategia de testing.

Al evaluar el modelo se encontró un incumplimiento con el criterio “Existe un plan de capacitación para la formación del personal en pruebas” debido a que este tema no fue tenido en cuenta dentro de la propuesta.

Se encontró que el equipo desarrollador no conocía todas las pruebas propuestas, lo cual presentó un retraso en los tiempos mientras se nivelaban los conocimientos, surgiendo como nueva idea de mejora contar con un repositorio de conocimiento donde se homologuen los conceptos y generen habilidades en cada una de las pruebas con una metodología didáctica tipo taller práctico que permita al equipo desarrollador alcanzar un grado mínimo de competencias aptitudinales antes de iniciar un nuevo proyecto.

La ejecución del proyecto piloto tenía una duración proyectada de 3 meses (6 sprints) y fue ejecutado en ese tiempo, el cual tenía como objetivo apoyar la labor comercial de la compañía mediante la presentación de un programa claro y específico para cada cliente trimestralmente y como alcance facilitar la interacción con el cliente ofreciendo información en tiempo real, facilitar la experiencia de uso en el acceso a la información, realizar gestión del conocimiento del proceso de negocio y garantizar la calidad de la información.

Se aplicaron las 4 categorías con sus tipos de pruebas definidas (8 tipos de pruebas en total) con una estimación de 50 horas para pruebas, lo que en un principio generó resistencia del equipo al requerir más tiempo para invertir en la ejecución de la estrategia de testing en comparación con la definida anteriormente, pero que al final del sprint se percibió por parte del equipo de desarrollo mejores resultados en la entrega de la funcionalidad.

Desde el inicio del sprint se tenía definido el catálogo de las herramientas a implementar según el tipo de pruebas (Tabla 10), lo que generó mayor eficiencia y disminuyó los tiempos que anteriormente se invertían en la búsqueda de las mismas cuando no se había trabajado algún tipo de las pruebas planteadas anteriormente, además de resaltar la importancia de que el experto en desarrollo con bagaje en testing participe en la estimación de tiempos para definir la duración del proyecto.

Tabla 10. Herramientas por tipo de pruebas

Tipo de prueba	**Herramienta**
Unitaria	Junit
Integración	Wiremock
Carga	Jmeter
Estrés	Jmeter
Seguridad	OWASP
Rendimiento	Jmeter
Funcionales	Cucumber
Aceptación	Serenity

Fuente: Elaboración propia

Se evidencia en el análisis inicial que el tiempo utilizado para encontrar soluciones a los errores repetitivos o similares era un gran desperdicio en la estrategia actual a lo que el modelo resuelve esta situación, obligando al equipo desarrollador a dejar evidencia de los errores y soluciones utilizadas para posterior consulta de los pares de otros proyectos o del mismo equipo. De igual manera el registro de éstas permite hacerle seguimiento a la estrategia y al proyecto por medio de distintas métricas fundamentales para la toma de decisiones, las cuales en la estrategia actual de la organización solo se utilizan a manera informativa, pero las encontradas en el proyecto piloto se desarrollan y analizan en el apartado 7.5

Se realiza la inspección por un par externo del proyecto con el objetivo de revisar porciones de código y velar que se esté cumpliendo con las buenas prácticas de desarrollo y de esta forma definir planes de mejora que permitan disminuir las brechas encontradas. Los resultados obtenidos de esta fueron producto del anexo 6 donde en la sección estética del código 3 de los 4 criterios evaluados fueron cumplido y se deja una recomendación la cual indica que "los nombramientos alineados al negocio deben ser más descriptivos, que al leer el código se esté expresando el negocio"; en cuanto a la sección arquitectura el 89% de los criterios se cumplieron, dos no aplicaban (Uso de procesamiento en batch y Manejo de buffers), en la sección manejo de errores el criterio hacer loggin de los errores no se cumplió, el apartado unitarias, integración y sistema se obtuvo el 100% de cumplimiento en los criterios. La inspección dio como resultado informe aceptado ya que las actividades del producto de trabajo tras la verificación han sido aceptadas sin ningún defecto encontrado.

Para validar las dos estrategias (estrategia actual de la organización y modelo propuesto) se realizó un cuadro comparativo frente a los productos de trabajo de CMMI-DEV en las áreas VAL, VER y PPQA (Tabla 11), observando que el modelo propuesto presenta una mejora a la estrategia actual haciendo uso de las buenas prácticas que se realizan en la organización y de las planteadas por CMMI-DEV.

Tabla 11. Correlación con las áreas de CMMI

Área de CMMI-DEV	Producto De Trabajo o Métodos	Estrategia Actual	Propuesta De Mejora (Modelo)
VAL	Requisitos y diseños de producto y de componente de producto	Se manejan en historias de usuario y se registra en el backlog del sprint	Se manejan en historias de usuario, queda registrado en el anexo 4 Informe de la estrategia sección medir
	Manuales de usuario	No aplica para la estrategia de testing, debido a que es producto del proyecto.	
	Entrega incremental del trabajo y del producto potencialmente aceptable	Sí, al estar inmerso en una metodología ágil, las entregas se dan en tiempos cortos y de manera incremental (Sprints)	Sí, además de estar inmerso en una metodología ágil, las entregas se dan en tiempos cortos(sprints) y las pruebas de la fase medir están diseñadas de manera incremental y se registran en el anexo 4
	Materiales de formación	No	No
	Documentación del proceso	Sí, se cuenta con un proceso documentado de consulta en la intranet la organización	Sí, se recomienda mantener la buena práctica de documentar y

			actualizar los cambios en el proceso en la intranet
	Entorno de validación	Cuenta con ambiente de pruebas	
	Procedimientos de validación	Sí, se ejecuta la estrategia de revisión par.	Sí, se ejecutan las pruebas de la categoría validación y se mantienen la inspección con la revisión par
	Informes de la validación	Sí, se deja registro de la revisión par, pero no existe registro de los resultados de las pruebas de validación cuando se realizan	Sí, se deja registrado en el anexo 4 los resultados de las pruebas de validación e inspección
	Resultados de la validación	Solo cuenta con resultados cualitativos de la inspección	Se registran los resultados cualitativos y cuantitativos en el anexo 4 sección medir y analizar
VER	Pruebas de cobertura de caminos	Sí, se tiene contemplada una cobertura del 79% en promedio	No, ya que durante el estudio se identificó que el porcentaje de cobertura no traduce que el código esté correctamente probado
	Pruebas de carga	El 74% no la realiza	Está definida como requisito del modelo
	Pruebas de estrés	El 92% no la realiza	Está definida como requisito del modelo
	Pruebas de rendimiento	El 92% no la realiza	Está definida como requisito del modelo
	Pruebas de aceptación	El 63% no la realiza	Está definida como requisito del modelo
	Lista de comprobación de la revisión entre pares	Sí, se tiene un formato estandarizado	Sí, se tiene un formato estandarizado anexo 4
	Resultados de la revisión entre pares	Solo cuenta con resultados cualitativos de la inspección	Se registran los resultados cualitativos y cuantitativos en el anexo 4
	Datos de la revisión entre pares		
	Informes de problemas.	Solo se determina la cantidad de defectos sin especificarlos	Sí, se registran en el anexo 4 sección mejorar
	Resultados de la verificación	Solo se determina la cantidad de defectos sin especificarlos	Sí, se registran en el anexo 4 sección mejorar y analizar
	Informes de la verificación	Se almacena en el sitio del proyecto	Sí, se registran en el anexo 4
	Informe de análisis	No	Sí, existen indicadores de gestión para evaluar la estrategia en todos los proyectos de la empresa, permitiendo conseguir tendencias del mejoramiento
PPQA	Informes de evaluación	Sí, se registra en un formato para la auditoria del proceso	Sí, queda registrado en el anexo 2 lista de verificación de la estratégica de testing
	Informes de no conformidad	Sí, se registra en un formato para la auditoría del proceso	Sí, queda registrado en el anexo 5, plan de mejoramiento
	Acciones correctivas		

	Informes de aseguramiento de la calidad	Sí, se registra en un formato para la auditoria del proceso	Sí, queda registrado en el anexo 2 lista de verificación de la estratégica de testing

Fuente: Elaboración propia

7.5 CONTROL Y SEGUIMIENTO

A continuación, se propone una serie de indicadores los cuales, mediante una expresión matemática, evalúan el comportamiento de una variables o característica de la estrategia de testing divididos en dos grupos (indicadores operativos y estratégicos), los cuales surgen a partir del marco teórico, diagnóstico y propuesta de modelo, estos además de contener su ficha técnica se ejemplifican para su mayor compresión. La tabla siguiente expresa un resumen de un proyecto como ejemplo para la construcción de los indicadores.

Tabla 12. Resumen proyecto ejemplo

Número de sprint	Tiempo dedicado a la solución del error (H=horas) por categoría de pruebas								Total Errores	Total Tiempo
	Unitarias		Integración		Sistema			Validación		
1	1,5 H	5 H	3 H	2,5 H	0,5 H	0,5 H	4 H	7 H	8	24 H
2	2,5 H	0,5 H	5 H	*	3 H	*	*	*	4	11 H
3	2 H	1,5 H	5 H	*	2 H	0,5 H	*	1 H	6	12 H
4	3 H	*	*	*	*	*	*	*	1	3 H
Total Errores	7 (2 abiertos)		4 (1 abiertos y critico)		6 (3 abiertos)			2	19	*
Total Tiempo	16 H		15,5 H		10,5 H			8 H	*	50 H

Fuente: Elaboración propia

Los indicadores de las Tabla 13 a la Tabla 18 pertenecen al grupo de indicadores operativos, debido a que controlan diversas variables de cada Sprint o ciclo del desarrollo. En primer lugar, tenemos la proporción de tiempo de esfuerzo por sprint (Tabla 13) este determina el tiempo que invierte el equipo de desarrollo en solucionar los errores dentro de un sprint.

Tabla 13. Ficha de indicador: proporción de tiempo de esfuerzo por Sprint

Nombre del indicador		Proporción de tiempo de esfuerzo por Sprint	
Meta	El promedio de tiempo invertido en la solución de errores no supere el 20% de las horas ejecutadas en el Sprint.		
Fórmula del indicador	Numerador	A	Sumatoria de los tiempos invertidos en la corrección de errores por Sprint
	Denominador	B	Total de horas del Sprint
	Fórmula		[∑ (A1+A2+ A3…An) / B] * 100

Fuente: Elaboración propia

Para este proyecto de ejemplo en cuyo Sprint 1 se ejecutan 2 historias de usuario, cada historia recorrió un ciclo de pruebas donde se tomó el tiempo invertido en la solución de cada error, dando como resultados 8 errores (Tabla 12), en el cual se toma la sumatoria de cada uno de los tiempos utilizados dando como resultado 24 horas en el Sprint 1. Si el tiempo determinado para el Sprint fue de 2 semanas (14 días por 9 horas laborales) en total se utilizó el 19% (24 horas/126 horas totales) del tiempo del Sprint, resultado del indicador dentro de la meta establecida. A diferencia del promedio de tiempo invertido en la solución de errores por Sprint el cual determina es el tiempo promedio que invierte el equipo de desarrollo en solucionar los errores dentro de un proyecto (Tabla 14).

Tabla 14. Ficha de indicador: promedio de tiempo invertido en la solución de errores

Nombre del indicador		Promedio de tiempo invertido en la solución de errores en el proyecto	
Meta	Se define según el histórico y el seguimiento a varios proyectos.		
Fórmula del indicador	Numerador	A	Sumatoria de los tiempos invertidos en la corrección de errores por Sprint
	Denominador	B	Total de Sprints en la ejecución del proyecto
	Fórmula		∑ (A1+A2+ A3…An) / B

Fuente: Elaboración propia

Para el ejemplo tenemos que en el Sprint 1 conto con un total de 24 horas, el Sprint 2 = 11H, el Sprint 3 = 12H, el Sprint 4 =3H; dividido el número de Sprint que tuvo el proyecto (4) dando como resultado 12,5 horas promedio invertidas en la solución de errores por Sprint.

La Tabla 15 corresponde al índice de errores reportados, este permite conocer la cantidad de errores en promedio durante un Sprint, para ello se toma la cantidad de errores que surgen en el primer Sprint (8), más los del segundo (4), tercero (6) y cuarto (1) y se dividen por el total de Sprints ejecutados (4), como resultado obtenemos 4,7 errores en promedio por Sprint.

Tabla 15. Ficha de indicador: índice de errores reportados en el proyecto

Nombre del indicador			Índice de errores reportados en el proyecto
Meta	Se define según el histórico y el seguimiento a varios proyectos		
Fórmula del indicador	Numerador	A	Sumatoria del número total de errores encontrados en cada Sprint.
	Denominador	B	Total de Sprints
	Fórmula		∑ (A1+A2+ A3…An) / B

Fuente: Elaboración propia

La Tabla 16Tabla 15 corresponde a la proporción de errores abiertos en el proyecto, este permite conocer la cantidad de errores que no fueron solucionados durante el proyecto, para ello se toma la cantidad de errores en estado abierto que surgen en cada Sprint (2), más los del segundo (1), tercero (3) y cuarto (0) y se dividen por el total de errores clasificados (19), por 100 como resultado obtenemos 31.5% errores por Sprint.

Tabla 16. Ficha de indicador: proporción de errores abiertos en el proyecto

Nombre del indicador	Proporción de errores abiertos en el proyecto
Meta	0% errores abiertos

Fórmula del indicador	Numerador	A	Sumatoria del número total de errores en estado abierto en cada Sprint.
	Denominador	B	Total de errores clasificados
	Fórmula		∑ (A1+A2+ A3…An) / B * 100

Fuente: Elaboración propia

La Tabla 17Tabla 15 corresponde al porcentaje de errores críticos abiertos, los cuales indican que existen afectaciones en la calidad del proyecto se toma la cantidad de errores abiertos y críticos del primer sprint (0), más los del segundo (1), tercero (0) y cuarto (0) y se dividen por el total de errores (19), multiplicado por 100, dando como resultado 5% , el cual se encuentra por fuera de la meta propuesta el cual requiere ser analizado e intervenido de manera prioritaria.

Tabla 17. Ficha de indicador: Proporción de errores críticos abiertos en el proyecto

Nombre del indicador			Proporción de errores críticos abiertos en el proyecto
Meta	0% de errores críticos abiertos en el proyecto		
Fórmula del indicador	Numerador	A	Sumatoria del número total de errores en estado abierto clasificados como críticos en cada Sprint.
	Denominador	B	Total de errores clasificados
	Fórmula		∑ (A1+A2+ A3…An) / B *100

Fuente: Elaboración propia

El indicador de efectividad de las pruebas según la categoría (Tabla 18) permite ver el grado de cumplimiento del objetivo de las pruebas de cada categoría (unitarias, integración, sistema, validación), para ello se debe conocer el número de errores clasificados por grupo de pruebas y al ser un modelo escalonado parte de la premisa que a medida que se avanza en las pruebas se tiene que ir disminuyendo el número de errores, siempre y cuando éstas sean realizadas correctamente. Para el ejemplo vemos que la efectividad de las pruebas unitarias sería 8 divido 19 (8+4+6+1) dando como resultado una efectividad del 42%, la efectividad de las pruebas de integración sería 4 sobre 11 (4+6+1) = 36%; la efectividad

para las pruebas de sistema sería 6 sobre (6+1) = 85%; indicando que solo las pruebas de sistemas se encontraron dentro de la meta establecida y es de anotar que el indicador no permite conocer la efectividad de las pruebas de validación debido a que su construcción siempre daría un porcentaje igual a 100, debido a esto su efectividad es evaluada con la inspección.

Tabla 18. Ficha de indicador: efectividad de pruebas según la categoría

Nombre del indicador		Efectividad de pruebas según la categoría	
Meta	Efectividad superior al 85%		
Fórmula del indicador	Numerador	A	Número de errores detectados en la categoría unitarias
		B	Número de errores detectados en la categoría integración
		C	Número de errores detectados en la categoría de sistema
		D	Número de errores detectados en la categoría de validación
	Denominador	A	Número de errores detectados en la categoría unitarias
		B	Número de errores detectados en la categoría integración
		C	Número de errores detectados en la categoría de sistema
		D	Número de errores detectados en la categoría de validación
	Fórmula		Unitarias: A / (A+B + C +D) * 100
			Integración: B / (B + C +D) * 100
			Sistema: C / (C +D) * 100

Fuente: Elaboración propia

A continuación se presentan los indicadores que hacen parte del grupo estratégico, su generación es producto de la estrategia de testing global, su análisis permite la toma de decisiones a nivel estratégico de la organización.

La Tabla 19 corresponde a la ficha del indicador "proporción de satisfacción global", la cual da como resultado la percepción del cliente acerca del producto desarrollado, a partir de la encuesta realizada a todos los *stakeholders* del proyecto 2 meses después de la entrega, donde se le pregunta sobre la satisfacción del producto entregado.

Tabla 19. Ficha de indicador: proporción de satisfacción global

Nombre del indicador		Proporción de satisfacción global	
Meta	Porcentaje de satisfacción del producto entregado mayor a un 80%		
Fórmula del indicador	Numerador	A	Número total de personas que presenta satisfacción con el software
	Denominador	B	Número total de encuesta realizadas
	Fórmula		A / B *100

Fuente: Elaboración propia

El indicador de costo de solución (Tabla 20) muestra los costos en valor monetario de la solución de los errores durante el desarrollo, que permite realizar un análisis comparativo frente al valor del proyecto. Éste se calcula teniendo como base en qué fase se detectan los errores, los cuales poseen un factor constante a lo que muestra que los costos son mayores cuando el producto ya está entregado, teniendo como premisa que el costo por cada fase es el triple del anterior. Como ejemplo partimos de que los errores fueron encontrados en construcción, lo que indica que el valor de referencia sería 4 por un total de 50 horas invertidas en la solución. Si el valor de hora desarrollador es de $20.000, el costo total de solución de errores sería 9'000.000.

Tabla 20. Ficha de indicador: costo de soluciones de errores del proyecto

Nombre del indicador	Costo de soluciones de errores del proyecto

Fórmula del indicador	Numerador	A	Número de horas en solución de los errores del proyecto
		B	Valor costo hora desarrollador
		C	Factor constante que aumenta según la fase del desarrollo en la cual se encontró los errores: 1 = Fase de requisitos. 3 = Diseño 9 = Construcción 27 = Producción
	Fórmula		(A * B * C)

Fuente: Elaboración propia

A continuación, se consolidan los datos del proyecto piloto en la Tabla 21 insumo para la construcción de las métricas propuestas.

Tabla 21. Resultados del proyecto piloto

Número de sprint	Tiempo dedicado a la solución del error (H=horas) por categoría de pruebas								Total Errores	Total Tiempo
	Unitarias		Integración		Sistema			Validación		
1	2H	1 H	1 H		1 H	0.5 H	0.5 H	0.5H	1.5 H	8	8 H
2	1 H	1.5 H	1 H	0.5	0.5 H		0.5 H	1 H	1 H	8	7 H
Total Errores	4		3		7			2	16	*
Total Tiempo	5.5 H		2.5 H		4.5 H			2.5 H	*	15 H

Fuente: Elaboración propia

El proyecto piloto se ejecuto en 2 sprints cuya duración fue de 4 semanas equivalente a 36 horas laborales donde el promedio de horas utilizadas para la solución de errores fue del 41% (15 horas/ 36 horas*100), el cual no cumple con la meta estipulada para el indicador; esto debido a que los desarrolladores requirieron tiempo para entender los nuevos tipos de

pruebas definidos; se invirtió en promedio 7.5 horas (15 horas /2 Sprints) por sprint para la solución de errores, se tuvo 8 errores (16 errores/ 2 sprints) promedio por sprint, de los cuales se obtuvo 0% (0 errores abierto/16 errores) de errores abiertos y 0% (0 errores abierto críticos /16 errores) de errores abiertos y críticos.

La efectividad de las pruebas unitarias fue de 25% (4 errores en pruebas unitarias /4 errores en pruebas unitarias+ 3 errores en pruebas de integración + 7 errores en pruebas de sistema + 2 errores en pruebas de validación); 25% para las pruebas de integración (3 errores en pruebas integración / 3 errores en pruebas de integración + 7 errores en pruebas de sistema + 2 errores en pruebas de validación);77% para las pruebas del sistema(7 errores en pruebas de sistema /7 errores en pruebas de sistema + 2 errores en pruebas de validación), lo que muestra que ninguno de los tipos de pruebas cumplieron con la meta establecida en la métrica, por tal razón se hace necesario iniciar un plan de mejoramiento con el fin de subsanar esta desviación del indicador que va enfocada a la capacitación en los distintos tipos de pruebas por categoría diferentes a las unitarias, ya que estas últimas son de las que más tienen dominio.

8 DISCUSIÓN DE RESULTADOS

La discusión de los resultados que a continuación se presentan se dividen en 2 apartados (8.1 Diagnostico actual; 8.2 Propuesta de mejora) que dan cuenta de los contrastes entre distintos autores y la posición del investigador frente a los resultados encontrados.

8.1 DIAGNÓSTICO ACTUAL

Se evidencia el gran número de estudios sobre tipos de pruebas y terminología alrededor del tema de investigación, donde confluyen en la importancia de la calidad de los productos para las empresas de software. Las pruebas, aunque son una herramienta que permite identificar fallas a tiempo, no garantizan que existan errores ocultos y en muchas ocasiones probar de extremo a extremo todas las entradas y condiciones en tan corto tiempo es algo imposible y dispendioso (Mera Paz J. A., 2016).

Teniendo en cuenta que la estrategia de testing empleada en la empresa está implícita en el proceso de desarrollo de software, es el mismo equipo de desarrollo quien ejecuta ambas tareas, esto implica que la organización debe contar con personal en sus equipos que tenga el conocimiento en los diferentes lenguajes, lógica computacional y algoritmia, todo esto inmerso dentro de una metodología ágil, lo cual exige que los desarrolladores ganen en experiencia con cada Sprint y cada proyecto que ejecuten, por lo tanto asuman el control de la calidad para garantizar la satisfacción del cliente (Bagarotti A, Meneses Abad, & Arias Guerra, 2013). Además, este rol (tester), como indica Perez-Lamancha, debe tener un pensamiento destructivo para la búsqueda de fallos (Perez Lamancha, 2006), y se debe tener en cuenta un plan detallado de capacitación como apoyo a las personas para que de esta forma puedan fortalecer sus habilidades en el rol de tester, los cuales en la estrategia actual de la organización nos e tienen contemplados.

Perez-Lamancha en su trabajo resalta la ventaja que tiene que los encargados de las pruebas sean un grupo diferente al del equipo desarrollador y en lo posible sea una organización externa, esta independencia del creador y el probador evita sesgos y afianza la rudeza de las pruebas (Perez Lamancha, 2006); por la misma línea Rojas-Montes et al consideran que el

equipo de testers no deberían ser los mismos desarrolladores, debido a que en su mayoría no cuentan con suficientes fundamentos teóricos y tienen un total desconocimiento de las técnicas de pruebas formales (Rojas Montes, Pino Correa, & Maur, 2015). Llama la atención que estas consideraciones estén alineadas con los resultados encontrados en la pregunta ¿considera usted que debe existir un área de talento humano especializado para las pruebas? a lo que el 86% de la muestra respondió que debería existir, quizás revelando que los desarrolladores de la empresa consideran que sus habilidades de pruebas aun no son los suficientemente fuertes para llevar a cabo las mismas y por eso la intención de contar con un área que tenga talento humano con el bagaje en todos los tipos y categorías de pruebas existentes para desligar esta función del desarrollador.

En contra a lo anterior, el poseer una estrategia independiente donde el equipo de desarrollo sea diferente al que ejecuta las pruebas tiene la desventaja de olvidar aspectos importantes a la hora de hacer las pruebas, debido al poco conocimiento y dominio del negocio, lo que genera sobrecostos en tiempos por la adquisición del conocimiento (curva de aprendizaje) y el que requiere cada equipo de manera independiente. Teniendo en cuenta que 8 de cada 10 desarrolladores consideran que debería existir un área independiente para las pruebas, Fernández-Sanz explica que este pensamiento es debido a que suele ser menos motivante ya que se considera una actividad poco creativa en comparación con el desarrollo (Fernandez Sanz, 2005).

Lo anterior expuesto demuestra que aunque ambas opciones son válidas y cada una presenta condiciones específicas, se podría contar con talento humano especializado dentro del equipo de desarrollo a cargo de la misma gerencia del proyecto y un par externo para la revisión experta del modelo de negocio.

Con respecto al diagnóstico del estado actual de la estrategia de testing en la empresa Ceiba Software, se concluye que en las categorías mencionadas, para el 100% de la muestra es de gran importancia realizar pruebas unitarias y además de ello en su mayoría la incluyen dentro de su ciclo de desarrollo, la cual permite validar la funcionalidad esperada según la

lógica de negocio, siempre y cuando sea evaluado el flujo normal y alterno de una aplicación y no solo se realicen por satisfacer un porcentaje de cobertura. Las pruebas de integración son indispensables, pero menos de la mitad la realizan.

Dentro de la categoría validación fueron consideradas indispensables para una estrategia de testing y que además son incluida en el ciclo de desarrollo, las pruebas funcionales y de aceptación. La mayoría de la muestra considera que una buena estrategia de testing debe contar con al menos un tipo de prueba de cada categoría.

Sobre la cobertura podemos decir que es el grado de profundidad que tiene testeada una funcionalidad, esta proporción se presenta de manera porcentual donde a valores más altos, mayor cobertura y más exhaustivas las pruebas, por lo tanto, menos posibilidad de error. Fontela en el 2013 afirmó que lograr alcanzar un 100% no es beneficioso frente al costo y por lo tanto es considerable llegar a un 80% o 90% de cobertura, valores similares fueron encontrado en el diagnóstico inicial donde el promedio del porcentaje de cobertura estuvo en 79,4%, no obstante se puede caer en el error de ejecutar pruebas sencillas que no generan valor solo por el hecho de obtener mayor cobertura (Fontela, 2013). Lo mencionado anteriormente coincide con la respuesta del encuestado E18 “en ocasiones se solicitan pruebas unitarias a métodos que no generan valor, todo para ganar un porcentaje de cobertura por encima del que ya hay en el proyecto”.

Dados los resultados de la matriz de Vester donde se encontró que uno de los desperdicios críticos fue el alto acoplamiento, Ponce et al afirma que cuando se cuenta con una gran cantidad de elementos gráficos (bloques, figuras, avisos, entre otros) que interactúan internamente entre sí y estos se encuentran en una misma capa visual provocan un alto acoplamiento, lo que implica que a la hora de modificar pequeñas líneas de código o agregar nuevos elementos genere un mayor esfuerzo de programación que muchas veces necesita de la revisión de posibles efectos colaterales en la capa visual, dichas modificaciones demandan tiempo y esfuerzo de desarrollo (Ponce A, López, Francisco A, & Almazar M, 2010), de esta misma forma Visconti y Astudillo afirman que cuando hay un

alto acoplamiento se generan problemas como la dificultad de entender cada dependencia o pieza cuando se aíslan, lo que hace difícil reutilizar código puesto que dependen de otras clases (Visconti & Astudillo).

Aunque la falta tiempo no surge como uno de los problemas críticos, sí presentó una alta frecuencia del mismo, lo que indica la importancia de incluir los tiempos requeridos para las pruebas en la planeación del proyecto, Dávila et al, en su trabajo indica que en la gestión del proyecto es recurrente observar que cuando hay demoras en desarrollo, se sacrifique el tiempo del equipo de pruebas, influenciado por la presión externa del cliente que llega como algo negativo para el equipo de pruebas (Dávila, García, & Cóndo, 2017).

Teniendo en cuenta que la empresa objeto en su modelo tiene integrado el uso de metodologías ágiles, este factor para Vaca et al considera que en ocasiones los periodos para el desarrollo al ser tan cortos (dos a cuatro semanas) se entrega un producto de software funcional, impidiendo la realización de las pruebas exploratorias hasta pruebas de regresión extensivas ya que requiere de mucho tiempo para su completitud (Vaca, P. A, Maldonado, C, & et al, 2013). Por la misma línea Lomprey y Hernández coinciden en que la presión de tiempo para las pruebas y la tentación de recortar en calidad cuando hay problemas de tiempo o de recursos, son las principales causas de fallos durante el desarrollo de software (Lomprey & Hernandez, 2008).

El proceso de pruebas está bien definido pero no se ejecuta apropiadamente. Es típico el caso de que si hubiera más tiempo, las pruebas serían más efectivas. Las pruebas están todavía basadas en especificaciones y el énfasis se pone en la prueba de los requisitos. Normalmente no hay informe definitivo al final de las pruebas sobre el estado del software. La conclusión general suele ser si el software está listo o no para producción (Fernandez Sanz, 2005).

8.2 PROPUESTA DE MEJORA

Este proyecto construyó un modelo de testing estándar, escalonado y riguroso, basado en la evidencia científica para el mejoramiento del producto final, lo cual concuerda con lo expuesto en el trabajo de Mera-Paz, (Mera Paz J. A., 2016) donde resalta que desarrollar un proceso de testing para la calidad de software sin un orden definido o sin conocimiento previo, dificulta la comprobación de la funcionabilidad y usabilidad de un producto de desarrollo de software y convierte la estrategia en barreras para el desarrollo del producto.

La puerta de entrada del modelo se diseñó desde la propuesta comercial del producto, esto asegurando que el experto en pruebas del equipo contempla los posibles tiempos para la solución de errores, ya que el conocimiento acerca de las pruebas de software, habilidades y experiencia es vital para la calidad y el alcance de los objetivos planteados, sino dicho desconocimiento en fases iniciales ocasiona la omisión de los posibles errores que puedan llegar a ser críticos dentro del sistema implementado, tiempos de entrega y satisfacción final del cliente (A, Ossaba, & Velásquez Isaza, 2012), lo anterior se deja plasmado en el apartado definir del informe de la estrategia, procedimiento que no se tenía tan visible en la estrategia actual de la organización, la cual se identificó como una oportunidad de mejora para convertirlo en una fortaleza del modelo propuesto.

Durante la fase de medir, cuyo fundamento es el ciclo de pruebas presentado de manera lógica y gradual por categorías de pruebas, coincide con el estudio de Mera-Paz, el cual realza la importancia de realizar pruebas tempranas para identificar errores rápidos que ahorren recursos y esfuerzo del talento humano, así como la utilización de un modelo en "V" donde de manera escalonada se prueben las etapas del proyecto (Mera Paz J. A., 2016), compartiendo similitud con Rojas-Montes et al quienes reconocen que es necesario el diseño del plan de pruebas y crear conciencia de la diferencia que existe entre planificar y diseñar pruebas (Rojas Montes, Pino Correa, & Maur, 2015), conocimientos compartidos con Lomprey y Hernández los cuales plantean una construcción de manera lógica y escalonada que permite que los errores detectados en el inicio del proceso de desarrollo de software son más fáciles de resolver y por lo tanto menos costosos (tiempo y esfuerzo

humano y financiero) que los que se detectan más adelante (Lomprey & Hernandez, 2008), El modelo presenta 8 tipos de pruebas las cuales se deben realizar de manera ordenada según su categoría (unitarias, integración, sistema y validación) siempre y cuando las características propias del proyecto lo permitan; en contraste con la estrategia actual de la organización donde el foco del ciclo de pruebas estaba dirigido a las pruebas unitarias y en una pequeña porción a otros tipos de pruebas.

El modelo al contar con un insumo previo donde están definidas las herramientas para cada tipo de pruebas disminuye los tiempos que emplea el desarrollador en la búsqueda de los mismos y cada uno de los resultado, hallazgos, defectos y soluciones se registran en un formato único que permite la centralización de la información para su posterior consulta, seguimiento y control. Eliminando la falta de documentación respecto a los resultados de la estrategia de pruebas en los proyectos la cual fue identificada como oportunidad de mejora.

El planteamiento de métricas dentro del proyecto para la toma de decisiones organizacionales derivadas de la ejecución de las actividades y la constante evidencia de los resultado de las acciones mediante informes, es una de las fortalezas que plantea esta propuesta de modelo, teniendo en cuenta lo indicado Justas-Muñez et al en su estudio evidenciaron que la ejecución de pruebas eran insuficientes e incluso nulas y por tal motivo la información que se reportaba era muy diversa y los informes para la gerencia no representaban beneficios en la toma de decisiones (Justiz Nuñez, Gomez Suarez, & Delgado Dapena, 2014). Por la misma línea Mera-Paz afirma que para la toma de decisiones y generación de planes de mejoramiento es fundamental partir de los requisitos, realizar análisis documental, identificar defectos y hacer pruebas funcionales y no funcionales (Mera Paz J. A., 2016). El resultado de esas métricas y documentos se traducen en incidentes que deben ser llevados a seguimiento y no deben ser perdido de vista, para lograr un aprendizaje organizacional al contener un repositorio de información referente a incidentes y soluciones para consulta, estudios de casos entre otros (Rojas Montes, Pino Correa, & Maur, 2015). Leon-Martinez et al resaltan que estas pruebas no pueden quedar a la deriva y su registro de resultados es lo que permite llevar constancia de lo hecho y

aportan valor a la calidad del software probado (Leon Martinez, Aguilar Garcia, Vega Morales, & Gomez Florez, 2013).

Expuesto lo anterior, el modelo, mas que cuantificar errores y tiempo en la solución de los mismos, diseña un formato donde permite especificar los errores encontrados, clasificarlos según su criticidad al evaluar el impacto y la urgencia de los mismos y por último dejar registrada la solución del error en el repositorio del proyecto, diferente a como se realiza en la estrategia actual donde muchas veces no queda la trazabilidad de los incidentes detectados y menos si aún no se le ha dado solución (errores en estado abierto).

Esta construcción de un histórico o repositorio de pruebas planteado en la gestión del conocimiento es vital para que los nuevos miembros de la organización alcancen rápido su nivel de aprendizaje al conocer dichos errores y sus alternativas de solución, evitando reprocesos y agilizando los desarrollos; debido a que al momento de enfrentarse nuevamente con el error similar cuenten con la información necesaria del cómo resolverlo, sin necesidad de indagar con un par cuyo problema similar haya sido superado y de esta manera se optimiza el tiempo de realizar una serie de intentos de solución (A, Ossaba, & Velásquez Isaza, 2012).

Como todo ciclo de mejoramiento es necesario verificar que las tareas que se están realizando cumplan una serie de estándares, por esta razón el modelo acoge de la estrategia actual la inspección, la cual permite revisar si los desarrollos cumple con los criterios de calidad definidos en un formato propuesto por el modelo y en segundo lugar esta la auditoría al proceso, la cual se mantiene como buena práctica de la estrategia actual, solo que el modelo estandariza el insumo en una lista de chequeo para la revisión del mismo.

Finalmente todas las desviaciones encontradas en cada una de las fases del modelo producto de una métrica fuera de las metas establecidas, un proceso de inspección no aceptable o hallazgos en la auditoría del proceso serán llevados a un plan de mejoramiento,

el cual vela porque esas oportunidades de mejora se materialicen en acciones concretas que subsanen lo encontrado.

Como producto final visualizar el mejoramiento de un proceso partiendo de la evidencia científica, como se realizó en esta investigación, resalta el hecho de incluir la gestión del conocimiento como buena práctica y base de la estrategia, alineado a lo que Febles-Estrada et al encontraron al realizar un estudio donde incorporaban asignaturas específicas de pruebas en el rol del desarrollador, para lograr competencias y habilidades en el rol de probador (tester), a lo que consiguieron ver niveles superiores de la calidad del software y disminución de errores en producción. (Febles Estrada, Capote Garcia, Velazquez Cintra, Delgado Martinez, & Calzadilla Diaz, 2011) (Soto Duran, Reyes Gamboa, & Jimenez Builes, 2017) alcanzando talento humano altamente calificado para llevar a cabo la estrategia de pruebas, que de manera per se mejore la calidad del producto final, logre una satisfacción en el cliente y mejore la reputación de la organización.

9 CONCLUSIONES

Referente al diseño e implementación del modelo integrador entre la metodología DMAIC y las áreas PPQA, VER y VAL de CMMI-DEV se resaltó la importancia de estandarizar el ciclo por categoría de pruebas de manera escalonada, así como el tipo de pruebas por cada una de ellas para corroborar y asegurar la calidad del producto de software, además de construir toda la estrategia de testing dentro de un ciclo DMAIC que permite de forma sistemática y sistémica buscar el mejoramiento del mismo a medida de que se pone en marcha en cada proyecto, disminuyendo los desperdicios que se presentan en cada ciclo.

Al comparar los resultados obtenidos entre el diagnóstico inicial y el modelo propuesto, se observó que ambas estrategias llevan registro del número de errores producto de la estrategia de pruebas, pero solo el modelo propuesto permite llevar registro de las alternativas y tiempo de solución además de clasificarlos según su criticidad, con el ánimo de que a futuro al presentarse nuevamente errores similares se le pueda dar solución de manera más rápida. Las métricas permiten la toma de decisiones organizacionales para el mejoramiento de la estrategia de testing, pasando de un modelo estático a uno propuesto de manera dinámico.

El foco de las pruebas en la estrategia actual se centra en las pruebas unitarias, las cuales se realizan de manera constante y numerosa, pero los demás tipos son considerados en menor grado, muchas veces por los cortos tiempos definidos en la propuesta comercial que no permiten realizarlas o por la falta de conocimiento de las mismas, a lo que la propuesta resuelve con un modelo escalonado y la gestión del conocimiento.

Se construyen indicadores que podrían generar valor y permitan de manera cuantitativa llevar el seguimiento del ciclo de pruebas y de la estrategia de testing, consolidándose en un grupo estratégico y otro operativo, este último aplicado en el proyecto piloto, demostrando que a partir del uso de los mismos se identifican nuevas oportunidades de mejora.

10 LIMITACIONES Y RECOMENDACIONES

10.1 LIMITACIONES

Debido a que los proyectos de desarrollo en su mayoría requieren de largos periodos para su ejecución el modelo propuesto solo se validó en un proyecto de corto tiempo (3 meses).

Debido al alcance y al calendario de ejecución de la investigación se proponen los indicadores basados en el marco teórico y los resultados encontrados, pero para su validación se requiere de tiempo adicional y ser medidos en varios proyectos.

Si bien el modelo propuesto permite una estandarización del proceso de testing, hay ocasiones en que un proyecto por sus características específicas no puede seguir la metodología propuesta de manera rigurosa y requiere ser adaptada.

10.2 RECOMENDACIONES

Se le recomienda a la organización hacer uso del modelo y de sus indicadores propuestos ya que va alineado con la mejora continua que tiene como requisito CMMI-DEV para sostener la certificación en nivel 5.

Se recomienda diseñar un plan de capacitaciones que sea conocido por toda la organización donde se incluyan temas en distintas pruebas y herramientas para su ejecución.

Incluir en la malla curricular de la maestría temas más prácticos que abarquen todos los distintos tipos y herramientas relacionadas con el proceso de calidad del software que le permita al estudiante poner en práctica el conocimiento empírico.

11 REFERENCIAS

A, L., Ossaba, P., & Velásquez Isaza, J. (2012). GESTIÓN DE CONOCIMIENTO: LA SOLUCIÓN PARA DISMINUIR EL REPROCESO EN LAS PRUEBAS DE SOFTWARE. Ing. USBMed, 3, 48-53.

Alvarez Baez, M. A. (2017). Desarrollo de una mejora en la metodologia V en la fase de pruebas de software implementado en el campo de la aeronautica.

Ardila Albarracín, C. A., Pino Correa, F. J., Pardo Calvache, C. J., & Merchán Paredes, L. (2014). MaTGeC: hacia un maco de trabajo para la gestión cuantitativa de procesos de desarrollo de oftware en pequeñas organizaciones. Revista tecnura, 18(42), 126-138.

Bagarotti A, Y., Meneses Abad, A., & Arias Guerra, Y. (2013). Experiencia durante la gestion de la calidad en proyectos que usan metodologias agiles. Revista de ingeniera, 20(3).

Brucker, A., & Julliand, J. (2014). Editorial for the special issue of STVR on tests and proofs volume 2: tests and proofs for improving the generation time and quality of test data suites.

Bueno, P. M., Crespo, A. N., & Jino, M. (2006). Analysis of an artifact oriented test process model and of testing aspects of CMMI.

Cardenas Ospina, L. A., & Rodriguez Beltran, J. A. (2011). Sistema de gestión de pruebas para productos de software.

Chanatasing Toapanta, H. M., Oña Sinchiguano, B. E., & Orbea Jimenez, E. M. (2017). Mejora de procesos de software. Revista publicando, 1(12), 68-88.

Chauhan, Y. (2015). Six Sigma in Project Management for Software Companies.

Chiuki, V., Rubinstein, V., Boria, J., Rubinstein, A., Baglietto, A., Andino, S., & Rocha, A. R. (2015). Una Experiencia de Implementación Multimodelo de Alta, Madurez con CMMI y MPS-SW en Sofrecom Argentina. CIbSE.

Dapozzo, G. N., Greiner, C., Irrazabal, E., Medina, Y., Ferraro, M., Lencina, A., . . . Mascheroni, A. (2016). Metodos y herramientas de estimación, gestión cuantitativa de proyectos, trazabilidad de requerimientos y entrega continua, orientados a la

mejora de la calidad del software. XVIII Workshop de Investigadores en Ciencias de la Computación, (págs. 651-656). Argentina.

Dávila, A., García, C., & Cóndo, S. (2017). Análisis exploratorio en la adopción de prácticas de pruebas de software de la ISO/IEC 29119-2 en organizaciones de Lima, Perú. Revista Ibérica de Sistemas e Tecnologias de Información.

Dustra, E. W. (1970). Notes on structure programing. Obtenido de http://www.informatik.uni-bremen.de/agbkb/lehre/programmiersprachen/artikel/EWD-notes-structured.pdf

Estayo, M. G., Dapozo, G. N., Cuenca Pletsh, L. R., Greiner, C. L., Medina, Y., Ferraro, M. d., . . . Pinto, N. (2012). Evaluación de calidad de ssoftware, formación de recursos humanos y políticas públicas para la promoción de la industria del software en la NEA. XIV Workshop de investigadores en ciencias de la computación. NEA Argentina.

Esterkin, V., & Pons, C. (2012). Analisis y evaluacion del MDD (Model Driven Software Development) desde la perpectiva del nivel 2 del CMMI DEV 1.3. xviii cONGRESO aRGENTINO DE CIENCIAS Y COMPUTACIÓN.

Febles Estrada, A., Capote Garcia, T., Velazquez Cintra, A., Delgado Martinez, R., & Calzadilla Diaz, R. (2011). Una experiencia novedosa para el testing desarrollada por un departamento de pruebas de software. Revista Cubana de Ciencias Informáticas, 5, 1-15.

Fernandez Sanz, L. (2005). Un sondeo sobre la práctica actual de pruebas de software en españa. Revista Española de Innovación, Calidad e Ingeniería del Software, 1.

Flebes Estrada, A., Capote Garcia, T., León Perdomo, Y., Velazquez Cintra, A., Delgado Martinez, R., & Calzadilla Diaz, R. (2011). Una experiencia novedosa para el testing desarrollada por un departamento de pruebas de software. Revista cubana de ciencias infromaticass, 5(2), 1-15.

Flores Montes, S., & LLamoca Quispe, E. B. (2017). Mejora de proecso basados en CMMI dev v1.3 para la gstion de proyectos en una empresa desarrolladora de software.

Fontela, M. C. (2013). Cobertura entre pruebas a distintos niveles para refactorizaciones más seguras. Universidad Nacional de la Plata.

Garza Ríos, R. C., González Sánchez, C. N., Rodríguez González, E. L., & Hernández Asco, C. M. (2016). Aplicación de la metodología DMAIC de seis sigma con simulación disccreta y tecnicas multicriterio. Revista de metodos cuantitativos para la economia y la empresa, 22(1), 19-35.

Gonzalez Palacio, L. (2009). Método para generar casos de pruebas funcional en el desarrollo de software. Revista de ingeniera de la ciudad de medellin, 8(15), 29-56.

Hetzel, W. (1984). The complete guide to software testing.

IEEE, ISO / IEC / IEEE 24765:2010. (2010). Norma internacional ISO / IEC / IEEE - Ingeniería de sistemas y software - Vocabulario. Obtenido de https://ieeexplore.ieee.org/document/5733835

IEEE. (2004). Guide to the software engirneering body of knowledge. 3.

Iñiguez Carchi, L. d. (2017). Desarrollo de una propuessta de mejora de procesos en el área de aseguramiento de la calidad basada en el Modelo de Capacidad de Madurez Integrada para la unidad de telecomunicaciones e información de la universidad Nacional de loja.

Iza Chorlango, R. (2012). Implementación de la metodología seis sigma en la empresa INVELIGENT. sangolqui.

Jimenez Bibian, O. P. (2017). Pruebas de caldad aplicadas al sitio web allison. Mexico.

Jimenez Puello, J. d., San feliu, T., & Calvo Manzano, J. A. (2016). Una aproximación basada en metamodelado del área de proceso de Validación del CMMI: Un caso de estudio. Revista Ibérica de sistemas e tecnologias de informacao(17), 26-40.

Justiz Nuñez, D., Gomez Suarez, D., & Delgado Dapena, M. D. (2014). Proceso de pruebas para producots de software en un laboratorio de calidad. Ingeniera industrial, 35(2).

Leaños Rodriguez, A. Y. (2018). Mejora en la curva de tiempo de ejecución de proyectos de software con la integración de herramientas para la genereacion automatica de casos de pruebas en dualbiz. Facultad de ingeniera de la compuación y telecomunicaciones.

Leon Martinez, N. E., Aguilar Garcia, J. L., Vega Morales, E. F., & Gomez Florez, C. G. (2013). Herramienta computacional para la dpocumentacion de pruebas de software enmarcado en actividades de investigacion. Scientia et technica, 18(4).

Lewis, W. E. (2004). Software testing and continuos quality improvement. (T. Ling, Ed.) 2(1), 10.

Lomprey, G., & Hernandez, S. (2008). LA IMPORTANCIA DE LA CALIDAD EN EL DESARROLLO DE PRODUCTOS DE SOFTWARE. Facultad de Ingeniería y Tecnología.

Magallanes de los rios, C. M., & Vega Marca, J. (2015). Implementación de CMMI nivel 5 en una empresa de desarrollo de software peruana. Facultad de ingeniera de sistemas y electronica.

Martín Córdoba, F. (2015). Testing ágil en las Empresas de Software del Cluster TIC Villa Maria.

Mera Paz, J. A. (2015). Analisis del proceso de pruebas de calidad de software. Ingenieria solidaria, 12(22).

Mera Paz, J. A. (2016). Análisis del proceso de pruebas de calidad de software. Ingeniería solidaria, 12(20), 163-176.

Mhammed Almakadmeh, A. (2017). The ISBSG Software Project Repository: An Analysis from Six Sigma Measurement Perspective for Software Defect Estimation. Journal of Software Engineering and Applications, 10, 693-720.

Mosquera Diaz, J., Ocampo Cortezz, O., & Garcia Monsalve, L. S. (2014). Funtional prototype of web information system to assist planning software projects, based on CMMI-DEV 1.2. Revista TECCIENCIA, 9(17), 67-77.

Myers, G. J. (2004). The art of software testing. 2(1).

Ocampo, J. R., & Pavon, A. E. (2012). Integrando la metodologia DMAIC de sies sigma con la simulacion de eventos discretos en flexsim. Panama.

Perez Lamancha, B. (2006). Proceso de testing funcional independiente. Universidad de la republica.

Ponce A, H., López, M., Francisco A, & Almazar M. (2010). DESARROLLO DE COMPONENTES DE SOFTWARE EN BASE AL PATRÓN DE DISEÑO MEDIADOR: EL CASO DEL ORGANIZADOR GRÁFICO INTERACTIVO. Congreso Iberoamericano de Informática Educativa, 1, 571-578.

Puello, O. (2013). Modelo de verificaccion y validacion basado en CMMI. Revista investigacion e innovacion en ingenierias, 1.

Rojas Montes, M. L., Pino Correa, F. J., & Maur, J. (2015). Proceso de pruebas para pequeñas organizacciones desarrolladores de software. Revita facultad de ingeniera, 24(39), 55-70.

Salazar Martínez, E. (2014). Propuesta de procedimiento para realizar pruebas de caja blanca a las aplicaciones que se desarrollan en lenguaje Python. 3C TIC, 3(2), 89-114.

Salazar Moreno, E., Mendoza Muñoz, I., Montoya Reyes, M., & Castañeda, A. (2018). Evaluacion de proyectos six sigma y modelo de referencia interactiva DMAIC para su finalizacion eficinete en empresa electronica. VIII CONGRESO-ARGOS LIBRO DE PONENCIAS.

Sanz, A., Saldaña, J., Garcia, J., & Gaitero, D. (2009). TestPAI: Un área de proceso de pruebas integrada con CMMI. Revista española de innovación, calidad e ingeniería del software, 4(4), 6-20.

Shirshendu, R., & Sujoy , S. (2015). To reduce defect in software development: a six sigma approach. 9th International Quality Conference.

Software Engineering Institute. (2010). CMMI para Desarrollo, Versión 1.3. Carnegie Mellon.

Sokovic, M., Pavletic, D., & Kern Pipan, K. (Noviembre de 2010). Quality improvement methodologies - PDCA cycle, RADAR matrix, DMAIC and DFSS. Journal of achievements in materials and manufacturing engineering, 43(1), 476-483.

Soto Duran, D., Reyes Gamboa, A., & Jimenez Builes, J. (2017). Aplicación de la Gestión de Conocimiento al proceso de pruebas de software. Ing. USBMed, 10, 6-13.

Srinivasan, K., Muthu, S., Prasad, N. K., & Satheesh, G. (2014). Reduction of paint line defects in shock absorber through Six Sigma. Procedia Engineering, 97, 1755-1764.

Tigueros Vargas, R. G. (2015). Propuesta de implementacion del area de procesos de validacion y verificacion de CMMI utilizando metodologias agiles en la consultoria de sistema ofisis.

Trujillo Casañola, Y., Febles Estrada, A., Garcia Rodriguez, A. M., Betancourt Rodriguez, Y., & Cao Terrero, G. (2011). Aseguramiento de la calidad del proceso y el producto basado en la gestion del conocimiento. Engineering for smart planet, innovation, information technology and computation. Medellin.

Vaca, P. A, Maldonado, C, & et al. (2013). est-Driven Development-Una aproximación para entender su utilidad en el proceso de desarrollo de Software. 2347-0372.

Villamizar Suaza, K., Tabares García, J. J., & Zapata Jaramillo, C. M. (2015). Mejora de historias de usuario y casos de prueba. Cuaderno Activa, 7, 41-53.

Visconti, M., & Astudillo, H. (s.f.). undamentos de Ingeniería de Software. Chile: Universidad Técnica Federico Santa María.

12 ANEXOS

Anexo 1. Instrumento diagnóstico

Señor(a) participante, el presente instrumento tiene el propósito de levantar la información relacionada con el proceso de testing (pruebas) en la compañía y hace parte de un proceso investigativo de la Maestría en gestión y desarrollo de proyectos de software de la Universidad Autónoma de Manizales.

La encuesta que está a punto de responder es de carácter ANÓNIMO y esperamos que sea contestada de manera honesta.

Con su participación no recibirá beneficios personales de ninguna clase. Sin embargo, se espera que los resultados obtenidos permitan mejorar los procesos institucionales.

Preguntas

1. ¿Cuáles son las tecnologías principales de su proyecto back/front?
2. ¿Cuál es el porcentaje de cobertura definido para el proyecto?
3. ¿Sabe usted para qué sirve el porcentaje de cobertura y porqué se definió ese valor para el proyecto?
4. Seleccione los tipos de pruebas que considera indispensables al momento de entregar un producto de calidad así actualmente no las lleva a cabo:
   Pruebas unitarias; Pruebas de regresión; Pruebas de Rendimiento; Pruebas de integración; Pruebas del sistema; Pruebas de desempeño; Pruebas de estrés; Pruebas de carga; Pruebas de volumen; Pruebas de múltiples sitios; Pruebas de compatibilidad; Pruebas de integración de datos; Pruebas Recuperación y Tolerancia a Fallos; Pruebas de Seguridad; Pruebas de aceptación; Pruebas funcionales; Pruebas de GUI; Pruebas de configuración; Pruebas de estilo; Pruebas de Usabilidad; Otro (Cual).
5. ¿Por qué considera que las pruebas seleccionadas son indispensables?
6. ¿Qué pruebas realiza usted como desarrollador en el proyecto?
7. ¿Indique en qué herramienta hace cada una de las pruebas que realiza?
8. ¿Cuáles de las pruebas que realiza actualmente no generan valor?, Explique.

9. ¿Cuáles son los problemas/ bloqueantes/ impedimentos más frecuentes con los que se encuentra al momento de realizar las pruebas?
10. ¿Cuenta usted con un ambiente definido para hacer las pruebas diferente al de producción? SI/ NO.
11. ¿Considera usted que el hecho de contar con talento humano especializado en pruebas mejoraría la calidad del software desarrollado por la empresa? SI/ NO, Explique su respuesta.
12. ¿Considera usted que el proceso de pruebas de software que ejecuta actualmente debe ser mejorado? SI/ NO, Explique su respuesta.

Anexo 2. Lista de verificación de la estratégica de testing

Lista de chequeo para la revisión de la estrategia de testing dentro del proceso de desarrollo de software

Datos básicos	
Objetivo del proyecto	
Tiempo de duración propuesto del proyecto	
Tiempo de ejecución real del proyecto	
Alcance	
Cantidad de sprints ejecutados	
Cantidad de historias de usuario	
Cantidad de tipos de pruebas realizadas	

Convenciones: **C:** Cumple con el criterio o estándar establecido; **NC:** No cuenta con el criterio o cumple parcialmente el estándar establecido.

Criterio	**C**	**NC**	**Comentarios**
Se tiene definido los objetivos del proyecto			
Se tiene un plan de pruebas con objetivos definidos			
Se tiene una estrategia de pruebas con objetivos definidos			
Se definen las herramientas para cada tipo de prueba			
Se ejecutan todas las pruebas definidas en la estrategia			
Se deja documentada la evidencia de los resultados de las pruebas			
Se prueban todos los escenarios que se espera que un usuario ejecute			

Se tiene definido un % de cobertura			
Se realizan pruebas unitarias			
Se realizan pruebas de integración			
Se realizan pruebas del sistema			
Se realizan pruebas de validación			
Se documentan los problemas del proyecto			
Se documenta la solución de los problemas encontrados			
Se realiza revisión por un par			
Se tiene documentado los hallazgos encontrados en la revisión por un par			
Existe un plan de capacitación para la formación del personal en pruebas			

Anexo 3. Matriz de dependencia e influencia

		Un alto acoplamiento en el sistema	Métodos anidados difíciles de mockear	Falta de conocimiento en pruebas de software	Falta de documentación del sistema	Falta de definiciones funcionales	Falta de tiempo para realizar las pruebas	No contar con un ambiente adecuado para	Desconocimiento del negocio	Exceso de pruebas	No contemplación de tiempos de pruebas en la propuesta comercial	TOTAL
		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	
Un alto acoplamiento en el sistema	P1	0	3	0	0	2	0	0	1	0	0	**6**
Métodos anidados difíciles de mockear	P2	3	0	0	0	0	0	0	0	0	0	**3**
Falta de conocimiento en pruebas de software	P3	0	0	0	2	0	1	0	0	0	0	**3**
Falta de documentación del sistema	P4	0	0	0	0	2	0	0	1	1	0	**4**
Falta de definiciones funcionales	P5	0	0	0	2	0	0	0	3	0	0	**5**
Falta de tiempo para realizar las pruebas	P6	3	1	0	0	0	0	1	0	3	3	**11**
No contar con un ambiente adecuado para las pruebas	P7	0	0	0	0	0	0	0	0	0	0	**0**
Desconocimiento del negocio	P8	0	0	0	3	3	0	0	0	0	0	**6**
Exceso de pruebas	P9	3	1	3	1	0	0	0	0	0	0	**8**
No contemplación de tiempos de pruebas en la propuesta comercial	P10	0	0	1	0	0	1	0	1	0	0	**3**
TOTAL		**9**	**5**	**4**	**8**	**7**	**2**	**1**	**6**	**4**	**3**	

Anexo 4. Informe de la estrategia

Definir	
Nombre del proyecto	
Objetivo	
Alcance	
Duración	
Sprints estimados	
Sprints reales ejecutados	
Duración estimada para pruebas	
Tipos de pruebas recomendadas	
Observaciones	

Medir (diligenciar por sprint)						
Número de sprint				**Descripción historia de usuario**		
Duración del sprint (Horas)						
Categoría	Tipo de prueba	Herramienta de prueba	Cantidad de errores encontrados	Cantidad de errores solucionados	Tiempo de corrección del error	Clasificación del incidente
Unitarias	Unitarias					
Integración	Integración					
Sistema	Carga					
	Estrés					
	Seguridad					
	Rendimiento					
Validación	Funcionales					
	Aceptación					

Clasificación de errores						
N° de defecto (Consecutivo)	Explicación del defecto (Descripción del defecto encontrado)	Criticidad del error			Solución (Explicación para solucionar el defecto)	Estado
		Impacto	Urgencia	Nivel		
1						
2						
3						
4						

Analizar		
Nombre del indicador	Valor del indicador	Análisis
Proporción de tiempo de esfuerzo por Sprint		
Índice de errores reportados en el proyecto		
Promedio de tiempo invertido en la solución de errores		
Efectividad de pruebas según la categoría		
Proporción de errores abiertos en el proyecto		
Proporción de errores críticos abiertos en el proyecto		

Anexo 5. Plan de mejora

Logo de la compañía	**FORMATO PLAN DE MEJORA**		
	Código:	**Versión:**	**Fecha:**

Fecha del plan	dd	mm	aaaa
Fuente*:			

**Tipos de fuentes: indicadores fuera de la meta, inspección, auditorías (PPQA)*

#	Oportunidad de mejora / Hallazgo	Acción de mejora a implementar	Responsable	Plazo (días)

Anexo 6. Inspección

Lista de chequeo para la revisión par dentro de la estrategia de testing

Datos básicos	
Nombre del proyecto	
Gerente del proyecto	
Tecnologías del proyecto	
Nombre del revisor par	
Fecha de la revisión	
Calificación de la inspección*	

Convenciones: **C:** Cumple con el criterio o estándar establecido; **NC:** No cuenta con el criterio o cumple parcialmente el estándar establecido.

Criterio	**C**	**NC**	**Comentarios**
Categoría Unitarias			
Uso de estructura AAA de pruebas unitarias			
Uso de estrategias de mocks en las pruebas			
¿La PU verifica el uso de los mocks antes del assert?			
¿La PU valida un único criterio de negocio?			
¿Hay posibilidad de bucle infinito?			
¿El número de parámetros utilizados coinciden con los de la rutina de la llamada?			
¡Se han usado adecuadamente todos los warnings y mensajes de información?			
¿Está el código documentado correctamente?			
¿En el código hay procedimientos apropiados?			
¿Los comentarios en el código son adecuados?			
¿Es una documentación clara y correcta?			

Categoría Integración			
Existen pruebas de integración			
Se hace uso de patrones de diseño de pruebas de integración			
Sin consumos mockeados			
Categoría Sistema			
Si el proyecto ya terminó al menos un release, debe tener pruebas de rendimiento			
Analizar la estrategia de seguridad del proyecto e identificar posibles riesgos			
Categoría Validación			
Se basa en guías de estilos del framework que utiliza (Angular tiene uno propio)			
Arquitectura			
No existen utilitarios o clases y métodos estáticos			
Uso correcto de pools para el manejo de conexiones con algún recurso (se cierran y liberan conexiones)			
Uso de procesamiento en batch (con la base de datos)			
Uso adecuado de patrones de diseño (Repositorios, Inyección de dependencias, Lambdas...)			
Estructura lógica del proyecto (Distribución de archivos, paquetes, recursos, etc)			
Revisión de implementaciones asíncronas			
Responsabilidades claramente definidas			
Principios SOLID			

(.NET) Implementa mecanismos para la liberación de recursos (IDisposable)			
No mezcla niveles de abstracción (Que las capas no implementen responsabilidades que corresponden a otra capa)			
Manejo de Buffers (archivos)			
Estética del código			
Uso adecuado de las convenciones de nombramiento			
Legibilidad del código			
Nombramiento alineado al negocio			
Idioma unificado			
Manejo de errores			
Se manejan dos tipos de excepciones, de negocio y técnicas			
No llevar errores técnicos al usuario			
Se hace logging de los errores			
No perder errores, no borrar la traza			

Califique la inspección según los siguientes criterios*

- Aceptado: las actividades del producto de trabajo tras la verificación han sido aceptadas sin ningún defecto encontrado.
- Aceptado con recomendaciones menores: las actividades del producto de trabajo tras la verificación han sido aceptadas, pero se han encontrado pequeños defectos que necesitan modificaciones menores para corregirlos.
- No aceptado: Las actividades del producto de trabajo tras la verificación realizada no han sido aceptadas, se requieren ajustes para una nueva revisión par en una semana.