UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE

"MODELO DE TRANSICIÓN DE DESARROLLO Y EJECUCIÓN DE PRUEBAS MANUALES A PRUEBAS AUTOMATIZADAS COMO ESTRATEGIA EN EL ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE"

FREDDY ANGELO VALDEBENITO ALARCÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Cecilia Reyes
Profesor Correferente: Luis Hevia

Noviembre - 2019

DEDICATORIA

Dedicado a mis padres Marcelino y Ester, quienes se han esforzado por darme la mejor educación y llenarme de buenos valores y a mis hermanos Giovanna, Claudio y Giancarlo, quienes han estado siempre cuando los he necesitado y me han entregado todo su apoyo durante todo este proceso.

AGRADECIMIENTOS

Primero que todo agradezco a la profesora Cecilia Reyes, quien se ofreció a ser mi profesor guía para el desarrollo de esta memoria y estuvo siempre disponible cuando necesité de su ayuda, puesto que, gracias a eso, pude finalizar este proyecto con las expectativas planteadas y los objetivos propuestos.

A mis padres y hermanos, que siempre me hicieron saber lo orgullosos que estaban de mí al estudiar Ingeniería Civil en Informática y eso me daba fuerzas para seguir, me daban ánimos y me instaban a siempre continuar a pesar de los momentos difíciles que, tal vez, todo alumno pasa en algún momento de su carrera universitaria. Sé que hicieron un gran esfuerzo por mantenerme en esta senda de la educación, que en algunas ocasiones se les hizo difícil, pero aun así hacían lo posible por mantenerme enfocado en mis estudios y darme la tranquilidad suficiente para no decaer en el camino, por lo que sé que esto los hace tan felices como a mí. Esto es de ustedes y para ustedes y, más que agradecerles por su apoyo, agradezco al destino por haberme dado la familia que tengo.

A mi compañera de vida, María Antonietta, que estuvo dispuesta a sacrificar nuestro tiempo juntos para poder dedicarme a realizar este proyecto. Su apoyo fue incondicional, me aportó ideas y estuvo siempre a mi lado durante todo el período que me tomó generar este documento, deseando tanto como yo, de manera desinteresada, que terminara este proceso de la mejor forma.

Finalmente, a mis compañeros y amigos que, a pesar de que no todos estaban físicamente a mi lado, de alguna forma me hacían sentir su apoyo y me instaban a avanzar con esta memoria.

RESUMEN

Resumen— La implementación de pruebas automáticas dentro de una organización de desarrollo de *software* no es una tarea fácil, puesto que malas decisiones al respecto pueden llevar a gastos innecesarios y pérdidas de tiempo de los recursos humanos, sobre todo cuando la cultura organizacional históricamente considera solo pruebas manuales. Tomando como base esta premisa, es que el presente documento propone un modelo de transición a pruebas automatizadas mostrando una serie de fases descritas de forma secuencial que tienen como finalidad obtener un conjunto de automatizaciones funcionales para un determinado proyecto o proceso. Los resultados obtenidos en una organización de servicios financieros permiten validar el modelo, mostrando sus beneficios tanto en tiempos de certificación como económicos, comprobados por los involucrados en el proyecto.

Palabras Clave— Aseguramiento de Calidad; Caso de Prueba; Prueba Manual; Prueba Automática; Modelo.

ABSTRACT

Abstract— The implementation of automatic tests within a software development organization is not an easy task, since bad decisions in this regard can carry out unnecessary expenses and wasted time of human resources, especially when the organizational culture historically considers only manual tests. Based on this premise, it is that this document proposes a transition model to automated tests showing a series of phases described secuentially that are intended to obtain to obtain a set of functional automations for a given project or process. The results obtained in a Financial Services organization showing benefits both in certification and economic times verified by the stakeholders in the project.

Keywords— Quality Assurance; Test Case; Manual Test; Automatic Test; Model.

GLOSARIO

AECL: Atomic Energy of Canada Limited

AGF: Administradora General de Fondos

ATDD: Acceptance Test Driven Development

BCH: Banco de Chile

BD: Base de datos

BDD: Behavior Driven Development

BSI: British Standards Institution

CMF: Comisión para el Mercado Financiero

FURPS+: Functional-Usability-Reliability-Performance-Supportability-Plus

HDD: Hard Drive Disk

IEEE: Institute of Electrical and Electronics Engineers

IEC: International Electrotechnical Commission

IRS: Inertial Reference System

ISO: International Organization for Standarization

ISTQB: International Software Testing Qualifications Board

KLOC: Kilo Lines Of Code

QA: Quality Asssurance

RAM: Random Access Memory

SQA: Software Quality Assurance

SQAE: Software Quality Assesment Exercise

SQuaRE: System and Software Quality Requirements and Evaluation

TDD: Test Driven Development

UFT: Unified Functional Testing

UI: User Interface

URL: Uniform Resource Locator

XML: Extensible Markup Language

XP: Extreme Programming

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

Desde que se concibe la idea de un desarrollo de software hasta que finalmente es liberado, existen muchas etapas que tienen como objetivo presentar un sistema que satisfaga de manera óptima la necesidad para la cual fue diseñada, dentro de las cuales, una de las que más destaca, es el Aseguramiento de Calidad. Parece un concepto muy abstracto cuando se habla de un *software*, sin embargo, están muy identificadas sus tareas y alcances mediante modelos que buscan dar la orientación adecuada para conseguir los mejores resultados. Sobre esta perspectiva se encuentran las pruebas manuales, las cuales buscan, mediante una serie de ejecuciones secuenciales, certificar que un determinado componente de sistema realiza efectivamente lo que debe hacer y de la forma que fue pensada. Sin embargo, cuando el aplicativo sufre modificaciones constantes, la realización de pruebas repetitivas por una persona resulta cansador y tedioso, provocando que el factor humano pueda ocasionar errores en una certificación mostrando fallas de sistema cuando en realidad no existen. Para mitigar esto es que existen las pruebas automáticas, que al ser programables evitan errores de falsos positivos y dobles interpretaciones de los resultados de las pruebas.

Este documento de memoria busca entregar al lector, a través de un modelo, los lineamientos correctos a ser considerados a la hora de pasar de pruebas manuales a pruebas automáticas, recomendando la mejor forma de identificar aquellos casos aptos para automatización, programación y ejecución y mantener un buen control de estos de tal forma que puedan ser utilizados múltiples veces mientras el aplicativo sufra cambios en su estructura. Para validar esta propuesta, cada punto mencionado se valida en una organización real y con resultados medidos empíricamente.

El presente documento se estructura de la siguiente forma: el capítulo 2 presenta el estado del arte, detallando algunos de los modelos de calidad más utilizados hasta la fecha. También se presenta lo más relevante con respecto a lo que son las pruebas de *software*, comenzando por los fundamentos de pruebas, continuando con lo que son las estrategias de pruebas conocidas, para finalizar con una descripción de lo que son las pruebas manuales y las pruebas automáticas. En el capítulo 3 se propone un modelo que permite una transición manual-automática mediante una serie de etapas dependientes una de la anterior que permite finalizar con pruebas automáticas listas y funcionales. En el capítulo 4 se describe la propuesta de modelo implementada en una empresa de servicios financieros, en donde se muestran los resultados obtenidos. Para terminar, en el capítulo 5 se presentan las conclusiones obtenidas a partir del trabajo realizado.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 CONTEXTO

En la actualidad, las empresas de todo el mundo consideran la calidad como una propuesta de valor en sus productos, la cual se traduce en un ahorro económico, aumento en el prestigio organizacional y en general, una mejora en la entrega de los productos o servicios a sus clientes. Sin embargo, muchas veces se desconocían los alcances que el concepto de calidad realmente abarcaba, quedando relegado a la interpretación subjetiva de los involucrados, sobre todo, en lo que a la industria de *software* se refiere. Para hablar de calidad de *software*, es necesario contar con una serie de parámetros que permitan cuantificar el valor que un producto debe alcanzar para satisfacer las necesidades del cliente, pero la problemática surge a la hora de generar métricas que permitan evaluar cuantitativamente cada característica de éste, debido a que la mayoría de éstas no son posibles de cuantificar fácilmente, y generalmente, se establecen de forma cualitativa. Es por este motivo que surge el concepto de Aseguramiento de Calidad del Software (a.k.a. SQA), que es un marco de referencia que abarca todas las actividades relacionadas con el aseguramiento de calidad durante el ciclo de vida del desarrollo y pruebas. Alrededor de este concepto, se han creado una serie de estándares que tienen como fin homogeneizar los métodos de evaluación de calidad durante todo el ciclo de vida del producto, entre los que se pueden destacar:

- **Normas ISO/IEC 25000:** Se crea en 2005 como resultado de la evolución de normas anteriores ISO/IEC 9123 e ISO/IEC 14598. También conocida como SQuaRE (*System and Software Quality Requirements and Evaluation*), tiene como objetivo principal especificar una serie de requisitos y evaluación de características de calidad que sirvan de guía para el desarrollo de software [18].

- **IEEE 730:** *Standard for Software Quality Assurance Plans*. Define la información que debe contener un plan de aseguramiento de calidad para su inicio, planificación, control y ejecución y la relación que tiene con otros procesos dentro de la organización [13].

- **IEEE 1012:** *Standard for Software Verification and Validation*. Es un estándar que tiene como propósito definir los procesos de verificación y validación de un producto de *software* en términos de actividades específicas y tareas relacionadas que permitan determinar si el desarrollo de una actividad determinada cumple con los requisitos de esa actividad y si el *software* satisface las necesidades previstas.

Este puede ser aplicado a un *software* que se está desarrollando, manteniendo o reutilizando [14].

- **IEEE 1061**: *Standard for a Software Quality Metrics Methodology*. Estándar que define una metodología para establecer requisitos de calidad e identificar, analizar y validar procesos y métricas de calidad que abarca todo el ciclo de vida del *software* [15].

- **ISO/IEC/IEEE 29119:** *Software and Systems Engineering - Software Testing*. Tal vez una de las más relevantes en el área de SQA. Es una norma creada con el propósito de unificar los estándares de BSI, IEEE e ISO/IEC sobre pruebas de software. Esta crea un estándar único y definitivo que recoge y estandariza actividades presentes en todo el ciclo de pruebas. Se compone de 4 partes: Conceptos y Definiciones, Procesos, Documentación y Técnicas de Pruebas [1].

Independiente del esquema, estándar o metodología que se utilice, existe una tarea esencial y fundamental para cualquier sistema y que es realizada en cualquier organización que posea un equipo de SQA: las pruebas de *software*, debido a que es la forma de verificar y garantizar la calidad de los productos desarrollados antes de realizar la liberación para el usuario final. Dependiendo del tipo de organización, se pueden identificar diversos tipos de pruebas que se pueden ejecutar en distintas etapas del desarrollo, sin embargo, las que poseen mayor relevancia debido a su transversalidad y generación de valor son las pruebas funcionales, puesto que están orientadas a emular el comportamiento de un usuario final en diversas situaciones dentro del aplicativo, esperando que este responda de acuerdo a un comportamiento esperado o definido con antelación por los dueños del producto para entregar la mejor experiencia. En otras palabras, son pruebas específicas y concretas que validan que el sistema haga lo que tenga que hacer sin errores. Es por esto que la ejecución de pruebas se vuelve imprescindible.

Estas pruebas, desde sus inicios, siempre fueron manuales, las cuales son definidas cuidadosamente basadas en un documento de requerimientos o manual de usuario y que genera como resultado un plan de pruebas y una matriz de casos de pruebas. Esta visión comenzó a tener sus primeras modificaciones con la aparición de *eXtreme Programming* [20] a finales de los 80’ y con ideas refinadas durante los 90’, metodología que hizo mucho hincapié en la automatización de pruebas mediante un proceso adaptativo y orientado a las personas, dando paso a la aparición de la filosofía ágil y realizando una fuerte crítica a las metodologías tradicionales (cascada, evolutivo, espiral, iterativo, etc.) mostrándolas como procesos pesados y burocráticos [20].

Hoy en día, no se pueden clasificar los conceptos de pruebas manuales y pruebas automatizadas dentro de un marco metodológico en específico ni evaluar si una forma es mejor que la otra, puesto que depende, en gran medida, de las necesidades de la organización. No obstante, lo vertiginoso de la evolución de la industria del desarrollo de *software* provoca que cada vez se requiera ejecutar más pruebas, en menor tiempo y con una mayor confiabilidad, por lo que la adopción de pruebas automatizadas se está tornando imperativo si se quiere entregar un producto con las condiciones óptimas para ser utilizado.

1.2 IDENTIFICACION DEL PROBLEMA

Hasta hace algunos años, el SQA durante el desarrollo de *software* era un tema poco explorado y casi relegado a la última etapa del proceso productivo, sin embargo, poco a poco fue tomando mayor importancia con el surgimiento de estándares que lo fueron posicionando como una de las labores esenciales a la hora de crear un sistema, surgiendo certificaciones e instituciones dedicadas a visibilizar la calidad como el punto más relevante al momento de entregar un aplicativo a un cliente. Fue el 2002, debido a la fundación de la *International Software Testing Qualifications Board* (ISTQB) [17], cuando comenzó a cobrar mayor protagonismo y desde entonces ha sido un concepto en constante evolución que crece tan rápidamente, que obliga a las organizaciones a estar en adaptación continua si quieren mantenerse en el mercado. Actualmente, el SQA no solo está presente en las empresas de desarrollo; esto ha adquirido una transversalidad tal, que al día de hoy se requiere un análisis de calidad dual, esto es, un área de SQA por el lado del proveedor de desarrollo y una por el lado del beneficiario del aplicativo: el primero se encarga de realizar pruebas orientadas a verificar el cumplimiento de los requerimientos solicitados por el cliente y el segundo a verificar que sus requerimientos fueron realmente cumplidos. Esto ha provocado que cada vez se necesite mayor cantidad de profesionales preparados y adaptados a las nuevas formas de realizar pruebas de *software*, como lo es complementar su actual proceso de pruebas manuales con las pruebas automatizadas, pero la reticencia al cambio de algunas organizaciones hace que el proceso de adaptación a las nuevas tecnologías sea un proceso que involucra una inversión demasiado alta comparada con los beneficios que esto aporta producido, principalmente, por el poco conocimiento que las empresas poseen respecto a esto, teniendo que recurrir a costosas auditorías que ofrezcan una posible alternativa de adaptación, proponiendo muchas veces, cambios culturales dentro de la organización, de paradigma y nuevas formas de trabajo que pueden tornar este proceso inviable.

Tal problemática se vería enormemente solventada si se internalizaran estos procesos, teniendo como actores de análisis a los principales involucrados: jefes de proyectos, analistas SQA y *testers*, puesto que tienen una visión más pulida de lo que existe y los objetivos a cumplir. No obstante, con esto sólo se tendría claro el qué se quiere lograr, quedando entrampados en el cómo.

Actualmente no existe una metodología que indique cómo proceder de la mejor forma en una transición de pruebas manuales a automatizadas, quedando relegado solo a información dispersa que se puede encontrar en internet, experiencias personales de los involucrados y centrándose solo en temas de costos, pero ¿son éstos los principales factores a considerar? ¿Podríamos, de alguna forma, prever que las herramientas escogidas son las que realmente necesitamos? Estas interrogantes que no es posible responder con seguridad, se podrían mitigar, en cierta medida, con un modelo de transición de desarrollo y ejecución de pruebas manuales a pruebas automatizadas como estrategia en el aseguramiento de la calidad del *software*.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Generar un modelo de transición que permita a las empresas adaptarse a las nuevas tecnologías de pruebas automáticas de la mejor forma posible, transformando su actual modelo, de manera progresiva, a un modelo de pruebas automáticas, siguiendo una serie de parámetros que aseguren un resultado exitoso sin poner en riesgo los procesos productivos y minimizando los riesgos económicos que esto conlleva, lo que aportará valor a todas las organizaciones de desarrollo de *software* que basan sus procesos de calidad en las mejores prácticas planteadas por el Comité Internacional de Certificaciones de Pruebas de *Software*.

1.3.2 OBJETIVOS ESPECÍFICOS

1. Descomponer en tareas específicas el proceso de generación y ejecución de pruebas manuales para el aseguramiento de calidad de un software y establecer y describir los hitos más importantes de este proceso desde la toma de requerimientos, hasta el paso a producción del aplicativo, lo que permitirá poner

foco en aquellos aspectos críticos de la organización durante el proceso de transición.

2. Descomponer en tareas específicas el proceso de generación y ejecución de pruebas automáticas para el aseguramiento de calidad de un software y establecer y describir los hitos más importantes de este proceso desde la toma de requerimientos, hasta el paso a producción del aplicativo, para compararlo con el estado actual de la organización que posee un modelo de pruebas manual y así establecer metas claras a alcanzar con la automatización.

3. Analizar las principales herramientas utilizadas para la creación de pruebas automáticas y generar un cuadro comparativo, con el fin de obtener una visión macro de las opciones disponibles en el mercado.

4. Realizar un cruce entre los hitos de pruebas manuales y automáticas, de tal forma de encontrar aquellos puntos en común y aquellos discordantes entre ambos modelos y generar conclusiones genéricas aplicables a todo tipo de organización.

5. Generar la pauta con los mejores consejos que permitan la transición de un modelo manual a un modelo automático óptimo, ordenado, paulatino y progresivo validado en una empresa dedicada a la Administración de Información Financiera.

CAPÍTULO 2: ESTADO DEL ARTE

2.1 CALIDAD DE SOFTWARE

La descripción de este concepto no es fácil, puesto que cuando se habla de qué es la calidad, se plantea desde el punto de vista de algo que se puede ver y evaluar, pero, en función de esto, ¿cómo es posible definir la calidad de algo intangible como un *software*? Se tiende a dejar mucho a la subjetividad, sin embargo, para la calidad del *software*, esto es algo que no debe dar cabida a ambigüedades, debido a que, si un equipo puede poner su foco en la calidad, se reduce el retrabajo, impactando en una rebaja en los costos y en una puesta en marcha al mercado más temprana de los productos. Para empezar, es importante entender lo que actualmente se conoce por calidad y sus alcances en la industria del *software* y para esto, es necesario comenzar analizando su definición más general citando lo que la RAE menciona al respecto, indicando que calidad es una *"propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor"* [27]. Ahora, también es necesario precisar lo que se entiende por *software*, en donde la RAE señala que es un "*conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas a una computadora"* [26], definición muy similar a la que entrega el estándar IEEE 610 [29] que lo define como "*programas de computador, procedimientos y posiblemente documentación asociada y datos pertenecientes a la operación de un sistema computacional*". Teniendo esto como base, podemos precisar que la calidad del software va más allá de la cantidad de defectos que un sistema pueda tener, lo que difiere completamente con lo que se creía en unos inicios con KLOC (*Kilo Lines Of Code*) [25] donde se evaluaba la calidad de acuerdo a la cantidad de defectos por cada mil líneas de código, sino que esta debe orientarse a la satisfacción de requerimientos bajo la mirada de un ente evaluador, con lo que podemos aceptar lo que nos menciona IEEE 610 [29] al respecto, donde indica que calidad de software es el "*grado en el cual un componente, sistema o proceso satisface requisitos especificados y/o necesidades y expectativas del usuario/cliente*". A modo general, es una definición que la resume de forma clara, sin embargo, es preciso ampliarla, puesto a que se debe contar con un conjunto de indicadores o factores comparativos que permitan evaluar de manera objetiva el grado de satisfacción que el producto entrega al usuario.

Es difícil desarrollar una medición con valor cuantificable que otorgue un indicador directo del atributo en estudio para un producto de *software*, debido a sus características únicas:7

- Es un producto abstracto e intangible que no está supeditado a un proceso de fabricación y sin precedentes anteriores que permitan su comparación.
- A diferencia de un producto tangible, el software con defectos no es rechazado, sino mejorado.
- La complejidad de mantención de un producto de software es mucho mayor, debido a que un error detectado en su desempeño implica necesariamente un análisis en el diseño inicial.
- No existe un proceso de fabricación en serie, puesto que es un producto realizado a medida, lo que dificulta predefinir estándares de control de calidad.

No obstante, existen una serie de modelos que se crean como consecuencia de la evaluación de un conjunto de métricas en diferentes etapas y que sirven para dar indicios de la solidez de una aplicación.

2.1.1 MODELO MCCALL

Este modelo presentado en 1977, comúnmente conocido como factores de McCall, Richards y Walters, muestra una serie de elementos claves a ser considerados al momento de evaluar un *software* desde el punto de vista del cliente que se centran en tres aspectos importantes: sus características operativas, su capacidad de revisión y su adaptabilidad con el entorno. Estos factores se resumen en la Tabla 1 [25].

Es importante que, para poder aplicar este modelo de manera correcta, se establezcan medidas que permitan eliminar la subjetividad a la hora de realizar la evaluación. Para esto, se sugiere considerar las características propias del *software* y la metodología utilizada para su desarrollo, tales como: ciclo de vida del producto, cantidad de liberaciones parciales antes de realizar la liberación final, tipo de entorno en cual se pretende funcione, público objetivo hacia donde está orientado su utilización, etc.

Cabe destacar que, a pesar de ser éste uno de los primeros modelos desarrollados, aún mantiene vigencia y muchos de los modelos y normas posteriores se basan en él, como la Norma ISO 9126 [3], por ejemplo.

Tabla 1: Factores de calidad de McCall
Fuente: Elaboración Propia.

Aspectos	Factores	Descripción
Operatividad	**Facilidad de uso**	Esfuerzo que se requiere para operar el software, proporcionar entradas y evaluar salidas y facilitar la familiarización con el usuario.
	Integridad	Grado para controlar el control de accesos de usuarios y proporcionar mecanismos de seguridad.
	Eficiencia	Cantidad de recursos utilizados en el procesamiento de información y cómputo de datos.
	Corrección	Grado de cumplimiento de las especificaciones del cliente de software.
	Fiabilidad	Grado de precisión, consistencia y exactitud con que el software cumple con su función.
Revisión	**Facilidad de prueba**	Esfuerzo requerido para someter al software a pruebas que verifiquen y validen sus funcionalidades.
Adaptabilidad	**Facilidad de mantención**	Esfuerzo requerido para detectar y corregir fallos en cuanto a simplicidad, modularidad y consistencia.
	Flexibilidad	Esfuerzo necesario para modificar un software en cuando a capacidad funcional y datos.
	Reusabilidad	Grado con que un software o parte de él puede reutilizarse en otra aplicación.
	Portabilidad	Esfuerzo necesario para transferir el software de un sistema a otro manteniendo su estructura y
	Interoperabilidad	Esfuerzo requerido para interactuar con otra aplicación.

2.1.2 MODELO FURPS+

Este modelo fue elaborado por Hewlett Packard y responde al acrónimo formado por sus 5 factores y el "+" que responde a temas de diseño (ver Tabla 2). Este modelo establece parámetros y restricciones medibles en cada una de las actividades de desarrollo de software en el que destacan dos características importantes si se decide adoptar como modelo de evaluación de calidad [25]:

1. No todos los atributos reciben la misma ponderación: puede darse más énfasis a los aspectos de seguridad cuando se evalúa funcionalidad, o tal vez el rendimiento en cantidad de transacciones por unidad de tiempo, o darle más relevancia a la

estética en cuando a usabilidad, etc. En síntesis, esto dependerá del foco que tenga el desarrollo.

2. Los atributos deben considerarse cuando comienza el desempeño.

Tabla 2: Factores y atributos de calidad FURPS+
Fuente: Elaboración propia

Sigla	Tipo de Requerimiento		Descripción
F	*Functionality*	**Funcionalidad**	Requerimientos que debería realizar el sistema: características y capacidades del programa, generalidades de las funciones, capacidades, seguridad.
U	*Usability*	**Facilidad de uso**	Esfuerzo para utilizar el sistema: factores humanos, estética, consistencia, documentación.
R	*Reliability*	**Confiabilidad**	Solidez y robustez del sistema durante la ejecución: recuperación, precisión, predicción.
P	*Performance*	**Rendimiento**	Velocidad y eficiencia en la utilización de recursos: velocidad de procesamiento, eficiencia, consumo de recursos, rendimiento efectivo total, productividad, tiempo de respuesta
S	*Supportability*	**Soporte**	Factores que se dan durante y después de la implementación: adaptabilidad, mantenimiento, compatibilidad, configuración, instalación.
+	*Plus*	**Implementación**	Restricciones a la construcción: recursos, lenguajes y herramientas, hardware
		Interfaz	Restricciones para la interacción con sistemas externos (no es GUI)
		Diseño	Restricciones para el diseño del sistema
		Física	Restricciones para el hardware utilizado

2.1.3 MODELO DROMEY

Este modelo propuesto por Roeb Geoff Dromey reconoce que, para evaluar la calidad de un *software*, es imprescindible adaptar el modelo a las características específicas del producto, planteando una especie de meta-modelo dinámico que permite evaluar distintas etapas del proceso de desarrollo como levantamiento de requerimientos, diseño e implementación [10]. Principalmente indica que las propiedades medibles de un producto están íntimamente relacionadas con los atributos de alto nivel que en su mayoría son difíciles de medir, por lo que es importante identificar esta relación para construir el modelo adecuado. Para realizar este cruce, Dromey propone un esquema relacional binario entre 3 entidades (ver Figura 1).

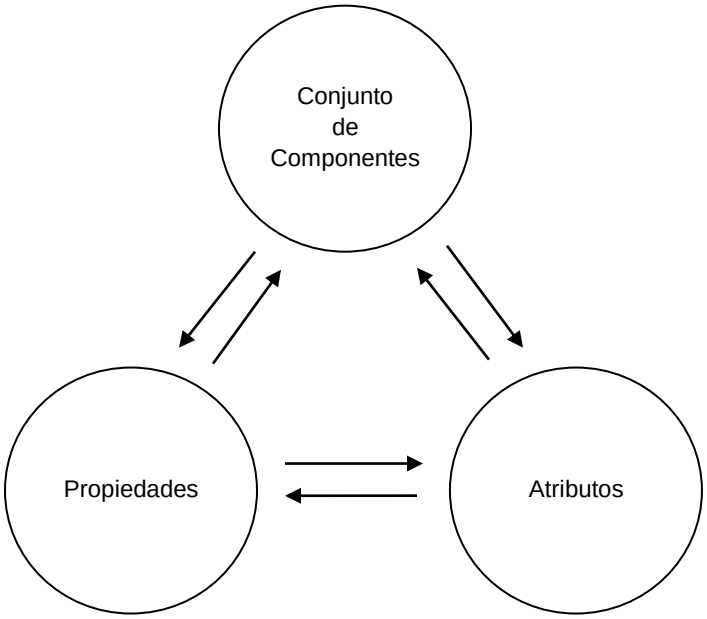


Figura 1: Esquema general del modelo de Dromey
Fuente: Exploración de modelos y estándares de calidad para el producto de software [23]

Este esquema señala que los atributos de alto nivel se obtienen con la construcción de componentes que representan un conjunto de propiedades basadas en 6 características principales: Eficiencia, Confiabilidad, Mantenibilidad, Portabilidad, Facilidad de Uso y Funcionalidad [2].

2.1.4 MODELO BOEHM

Este modelo propuesto por Barry Boehm extiende el modelo McCall agregando algunas características y se basa en definir la calidad en términos de atributos cualitativos, medidos a través de métricas [23]. Éste descompone las características de la calidad en 3 niveles: usos principales, componentes intermedios y componentes primitivos [9], donde cada uno se define a partir de criterios de evaluación (ver Figura 2).

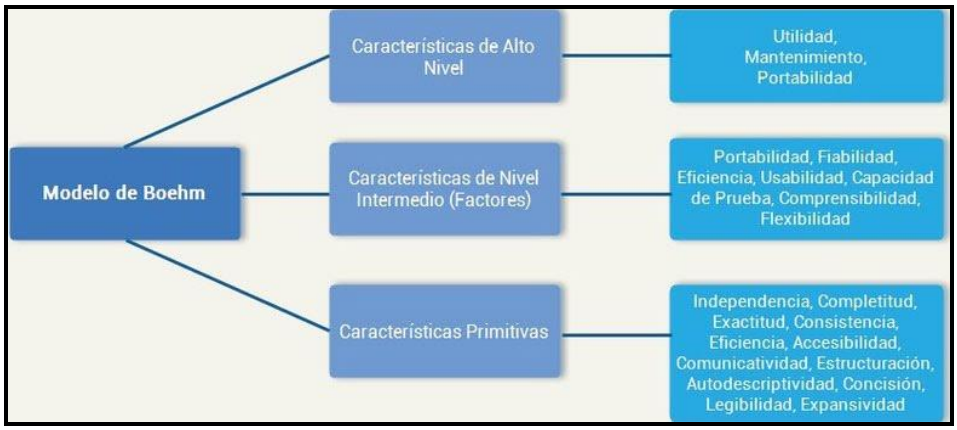


Figura 2: Esquema general del modelo de Boehm
Fuente: https://sites.google.com/site/moduloevaluacionred/modelo-de-calidad-boehm

La finalidad de este modelo es que el software [9]:

1. Cumpla expectativas del usuario
2. Sea eficiente en la utilización de recursos
3. Sea fácil de usar y de aprender
4. Sea bien diseñado, codificado, probado y mantenido

2.1.5 MODELO SQAE (Software Quality Assesment Exercise)

Este modelo basado en el de Boehm, McCall y Dromey y el estándar ISO/IEC 9126 fue desarrollado por MITRE Corportation y su solución se basa principalmente en asociar áreas de calidad con factores tangibles y mitigables. Su aplicación se centra en la asignación de porcentajes a los factores de calidad los cuales, a su vez, están constituidos por una serie de atributos asociados a métricas y medidas [23].

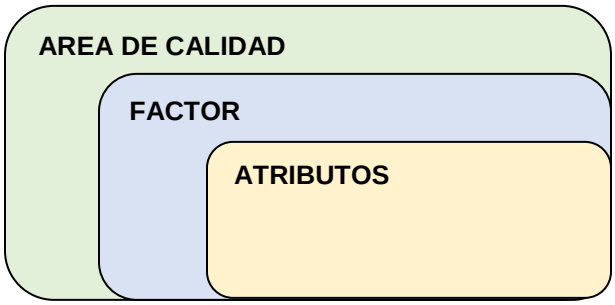


Figura 3: Jerarquía modelo SQAE
Fuente: Elaboración Propia

Cada área de calidad se utiliza para definir conceptos de riesgo que, para este modelo, son 4: Mantenibilidad, Evolución, Portabilidad y Descripción, los cuales están asociados a 7 factores menos abstractos: Independencia, Modularidad, Documentación, Auto descripción, Control de anomalías, Diseño simple y Documentación. La interdependencia área-factor se presenta en la Figura 4.

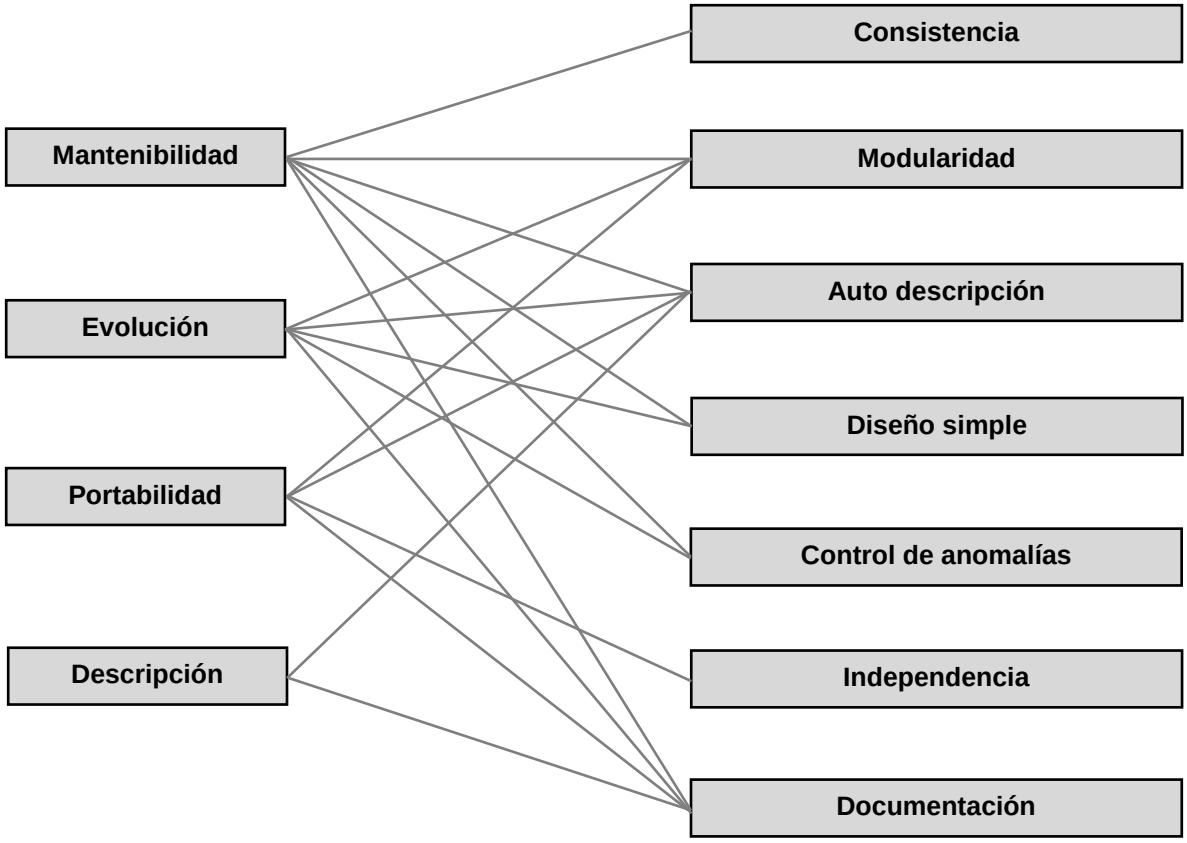


Figura 4: Interdependencia modelo SQAE
Fuente: Exploración de Modelos y Estándares de Calidad para el Producto de Software [23]

Para cada atributo dentro de un factor se define [9]:

1. Alcance de la evaluación
2. Dato del atributo
3. Criterio de evaluación
4. Contexto de evaluación

Una vez definido esto, se puede establecer un porcentaje a cada factor en base al riesgo asociado, obteniendo una traza tal como se observa en la Figura 5.

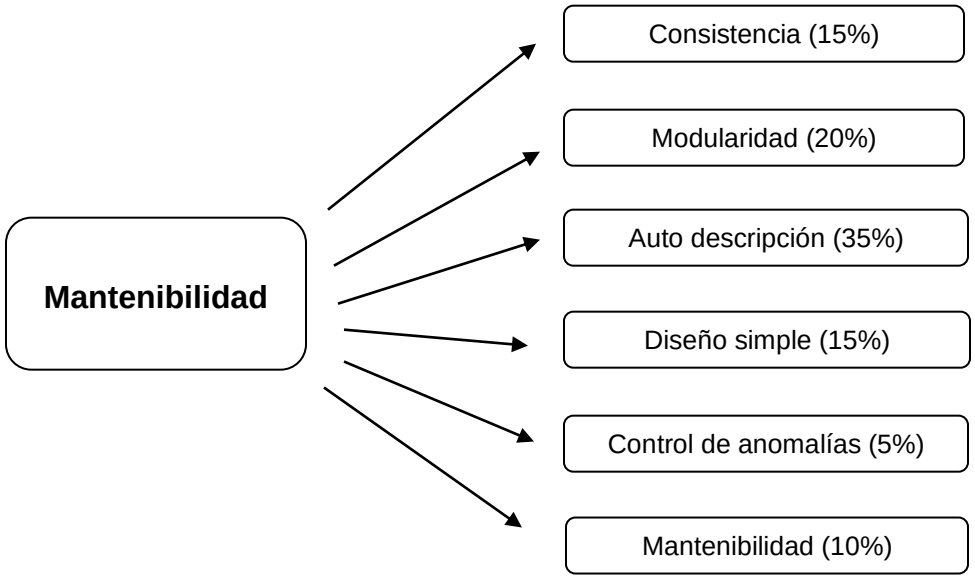


Figura 5: Porcentualidad área-factor
Fuente: Elaboración Propia

Cabe señalar que estos no son todos los modelos existentes, pero ayudan identificar el principal foco que estos persiguen: llevar a factores medibles aquellas características abstractas asociadas a un *software*.

2.2 FUNDAMENTOS DE PRUEBAS

2.2.1 ¿QUÉ SON LAS PRUEBAS Y POR QUÉ SON NECESARIAS?

Existen muchas definiciones respecto a lo que son las pruebas dependiendo del enfoque que estas tengan. Según Glenford Myers, científico informático, uno de los entendimientos que tienen los desarrolladores con respecto a las pruebas es: "*Las pruebas son el proceso de establecer confianza en que un programa hace lo que se supone que debe hacer*" [24]. Si bien, esta definición puede ser aceptada, es importante establecer de qué forma se produce la verificación que un programa o aplicativo cumple con los requisitos.

Edser W. Dijkstra (científico de las ciencias de la computación) tiene una visión clara respecto a las pruebas [16]: *"Las pruebas de software pueden probar la presencia de errores, pero no la ausencia de ellos",* muy similar a lo que señala Myers en su libro [24]: "*La prueba es el proceso de ejecución de un programa con la intención de encontrar errores*".

Considerando estas afirmaciones, podemos hablar de las pruebas como un proceso mediante el cual se pretende detectar defectos en un sistema con el objetivo de determinar que este cumple los requerimientos establecidos para el usuario. Esta definición le otorga suma importancia al proceso de pruebas, puesto que, un aplicativo que no funciona según lo previsto, puede dar lugar a muchos problemas tanto para un usuario como para el proveedor del aplicativo: pérdidas económicas, pérdida de tiempo, daños personales e incluso la muerte. Algunos antecedentes reales de defectos en el software que han producido consecuencias que se podrían haber evitado son:

- El proyecto espacial de la industria europea llamado Ariane 5 estalló 30 segundos después del despegue sobre la Guyana Francesa el 4 de junio de 1996. La comisión investigadora concluyó que el falló procedió del software del sistema de control y guiado (SRI): el programa recibió tanta información que enloqueció y transmitió información errónea al ordenador central, provocando una corrección de trayectoria imposible. Los SRI utilizados en el Ariane 5 eran los mismos utilizados en el Ariane 4 y como este cohete nunca presentó fallas, no repararon en probarlos una vez incorporados en el Ariane 5. A causa de esto, el informe recomienda incluir pruebas y ensayos en futuros proyectos [11].
- Durante la guerra del golfo en 1991, un conjunto de misiles Patriot estadounidenses no fue capaz de detectar e interceptar un misil Scud iraquí

entrante, lo que provocó que este alcanzara un cuartel del ejército matando a 28 soldados e hiriendo a otras 100 personas. Según los informes, la causa de esta falla fue un cálculo erróneo de tiempo de arranque, específicamente, se realizó un mal cómputo respecto a las décimas de segundo que medía el reloj interno de los misiles [8].

- El Therac-25 es uno de los casos emblemáticos que reflejan las consecuencias de depositar plena confianza en el software desarrollado sin ejecutar pruebas centradas en los usuarios finales. Esta fue una máquina diseñada para radioterapia por la empresa *Atomic Energy of Canada Limited* (AECL), como evolución a sus antecesores: Therac-6 y Therac 20. Debido a que gran parte del software venía de estas dos versiones anteriores y no habían presentado problemas, los analistas asumieron que no habría inconvenientes de este tipo, por lo que las pruebas de *software* quedaron de lado casi por completo. Una de sus funcionalidades era la de disparar haces de electrones de alta potencia para tumores más internos, interponiendo una placa metálica entre el paciente y el haz para evitar la exposición directa a la radiación. La falla se produjo cuando se activó un haz de alta potencia en lugar de una de baja sin que el sistema detectara la ausencia de la placa, exponiendo a sobredosis masiva de pacientes por lo menos 6 veces entre junio de 1985 y enero de 1987, llegando a lesiones graves e incluso muerte de pacientes [6].

2.2.2 PRINCIPIOS DEL PROCESO DE PRUEBAS

El Comité Internacional de Certificaciones de Pruebas de Software ha establecido 7 principios básicos que todo profesional en el área de calidad debe tener en cuenta a la hora de comenzar con un proceso de pruebas de software [19]:

1. **Las pruebas demuestran la presencia de defectos**

   Las pruebas pueden disminuir la cantidad de defectos presentes en un software, pero, a pesar de que no se encuentre ningún defecto en un proceso de pruebas, esto no asegura que el software carezca de ellos.

2. **No existen las pruebas exhaustivas**

   Es imposible probar todas las combinaciones posibles que un aplicativo puede tener, por lo que es mejor priorizar y centrarse en aquellos casos con mayor riesgo.

3. **Pruebas tempranas**

   La detección de defectos en una etapa temprana del proyecto se traduce en beneficios de tiempo, esfuerzo y económicos, por lo que es bueno comenzarlas los antes posible.

4. **Agrupación de defectos**

   Se postula que la mayor parte de los defectos se centra en una parte específica del *software*, por lo que es muy importante priorizar y centrar las pruebas en las partes sensibles.

5. **Paradoja del pesticida**

   Se plantes que las pruebas son un organismo vivo que debe actualizarse de manera constante a medida que avanza el proyecto, puesto que, si se realizan las mismas pruebas en distintas etapas del proyecto, llegará un punto en que se dejen de encontrar defectos.

6. **Las pruebas dependen del contexto**

   Las pruebas se ejecutan de forma distinta dependiendo del sistema en cuestión; se requieren distintos focos de atención, distintas habilidades y distintas estrategias para llevar a cabo un proceso de pruebas.

7. **La falacia de la ausencia de errores**

   Este es el principio básico y tal vez el más relevante a la hora de probar un aplicativo. Asegurar que un sistema no posee defectos acarrea una responsabilidad muy alta, puesto que, el hecho que un sistema haya pasado el proceso de pruebas de manera satisfactoria no asegura que quede exento de defectos.

2.2.3 CASOS DE PRUEBA

Un caso de prueba es la unidad básica y principal de todo proceso de certificación y debe responder a la pregunta: ¿Qué espero que haga el sistema? Está constituido por varios campos que tienen como finalidad proporcionar toda la información necesaria para ejecutar una determinada acción de acuerdo con lo planificado y evaluar el comportamiento de un componente, funcionalidad o módulo de una aplicación.

Los casos de prueba son elementos reutilizables; se ejecutan cada vez que un sistema o parte de él sufre modificaciones y en muchas ocasiones, no siempre por el que las elabora, por lo que es de suma relevancia que sean simples, sin ambigüedades, precisos, claros y en lo posible, independientes. Un caso de prueba diseñado de manera correcta, tiene mayores probabilidades de llegar a resultados confiables, aumentar el rendimiento del aplicativo y reducir costos en tres categorías principales:

1. Productividad, al reducir los tiempos para escribir y mantener los casos
2. Capacidad de prueba, al reducir los tiempos de ejecución
3. Fiabilidad, por otorgar la facilidad de realizar estimaciones más precisas y efectivas.

Los campos de un caso de prueba pueden variar de acuerdo con la estrategia de pruebas definida por el equipo, sin embargo, debe contener campos mínimos que se detallan a continuación [12]:

1. **ID**: Identificador único del caso de prueba. Usualmente suele ser un correlativo numérico o alfanumérico.
2. **Nombre del caso**: Etiqueta única que identifica el aspecto a probar.
3. **Descripción:** Resumen general de lo que se desea probar.
4. **Precondiciones:** Requisitos previos que se deben cumplir para obtener el resultado esperado.
5. **Datos de prueba:** Información necesaria para ejecutar el caso, como credenciales de acceso, URL, datos de BD, etc.

6. **Paso a paso:** Secuencia de acciones que permiten que el caso se ejecute de forma exitosa.
7. **Resultado esperado:** Objetivo esperado de la prueba.
8. **Estado:** Situación final luego de haber ejecutado el caso. Si bien no existe un consenso de cuáles son los estados correctos, los más comunes son los siguientes:
    - OK: Aquellos que finalizan de acuerdo con lo descrito en el resultado esperado:
    - NO OK: Aquellos que no se comportan de acuerdo con lo descrito en el resultado esperado.
    - Pendiente: Aquellos que aún no se han ejecutado.
    - Bloqueado: Aquellos que no se pueden ejecutar debido a algún defecto encontrado en los casos que lo preceden o alguna precondición.
9. **Resultado obtenido:** Se detalla el resultado finalmente obtenido del caso. Campo importante a la hora de reportar defectos.

Al conjunto de casos de pruebas se le denomina “Matriz de Casos de Pruebas” o “Matriz de Cobertura” y se construye, comúnmente, al inicio del proyecto bajo un modelo tradicional con base en los documentos de requerimientos, casos de uso y/o manuales de usuario.

2.2.4 TIPOS DE PRUEBAS

Existen dos grupos principales donde se pueden clasificar los tipos de pruebas que se pueden realizar para probar un sistema: pruebas funcionales y no funcionales. Las primeras pretender comprobar que lo que se está desarrollando funciona de acuerdo a las expectativas del cliente, mientras que las segundas consideran el comportamiento, es decir, cómo funciona el sistema [28].

Dentro de las pruebas funcionales existen:

- **Pruebas unitarias:** Su objetivo es realizar pruebas en módulos o unidades de software de forma independiente, analizando, por una parte, el código para corroborar que se están cumpliendo los requerimientos del producto, y por otro, la funcionalidad del componente. Por este motivo, es recomendable que estas pruebas sean realizadas por desarrolladores [28].

- **Pruebas de integración:** Se encargar de probar la interacción entre los distintos componentes o módulos y sus interfaces una vez que las pruebas unitarias resultan exitosas [28].

- **Pruebas de sistema:** Acá es donde se prueba si el producto cumple con los requerimientos establecidos para un usuario final. Es donde se utiliza una matriz de casos de prueba que cubra todas las funcionalidades del aplicativo [22].

- **Pruebas de regresión:** Son aquellas pruebas que tienen como objetivo verificar que las modificaciones que se realizaron no corrompen lo que ya está funcionando [7].

- **Pruebas de humo:** Su foco es comprobar el funcionamiento básico del sistema y que no se interrumpen los procesos básicos [7].

- **Pruebas de aceptación:** Para realizar un proceso de pruebas exitoso es importante que sean realizadas por el área especializada de QA, sin embargo, en las pruebas de aceptación son las únicas pruebas donde la persona que está a cargo es el cliente y se encarga, de acuerdo con su concepción, de comprobar que el producto que solicitó cumple sus expectativas [28].

Por otro lado, dentro las pruebas no funcionales existen [7]:

- **Pruebas de compatibilidad:** Comprueban el sistema en distintos entornos o sistemas operativos.

- **Pruebas de carga:** Verifican el comportamiento de un sistema dado un nivel de carga definido.

- **Pruebas de estrés:** Son pruebas de carga con un nivel mayor al esperado con el objetivo de romper esta capacidad.

- **Pruebas de usabilidad:** Principalmente se enfocan en evaluar la experiencia del usuario al utilizar las distintas funcionalidades del aplicativo.

2.3 ESTRATEGIAS DE PRUEBAS

Cada proyecto tiene sus particularidades, por lo que es importante saber integrar de manera correcta las técnicas de diseño de casos de prueba dependiendo de la metodología de gestión utilizada. La forma en que se realiza el proceso de ejecución de pruebas puede variar de tal forma que se ajuste al equipo, los recursos y tiempos estimados para la certificación del aplicativo.

En la actualidad existen dos grandes metodologías de gestión:

1. **Metodología Tradicional o Convencional**

   Las metodologías tradicionales dividen el proyecto en fases y actividades de trabajo rígidas y bien definidas que se deben realizar para crear un producto terminado en donde la culminación de una marca el inicio de la siguiente. Cada una de las actividades, acciones y tareas se encuentran dentro de una estructura que define su relación tanto con el proceso como entre sí [25]. Se trabaja principalmente con ciclos de prueba, donde cada ciclo genera defectos dentro del aplicativo y cuando son solucionados por parte de área de desarrollo, comienza el siguiente ciclo para verificar que los defectos fueron resueltos y que no existen nuevos.

2. **Metodología Ágil**

   La metodología ágil parte de la premisa del cambio: es difícil predecir los requerimientos del software que persistirán y cuales deberán sufrir cambios, por lo que propone que los procesos deben adaptarse a medida que avanza el proyecto, donde constantemente se va obteniendo retroalimentación del cliente, permitiendo que este evalúe regularmente el incremento del producto e influya en las adaptaciones del proceso que se realicen para aprovechar la retroalimentación entregada [25].

Una de las principales diferencias entre ambas, es que la primera tiene el foco en realizar una única entrega de un producto final terminado, lo que implica que los cambios o nuevos alcances que surjan por parte del cliente se realicen sobre un software funcionalmente apto para ser evaluado. Por otra parte, la metodología ágil apunta a realizar pequeñas entregas funcionales del aplicativo a medida que se avanza en el desarrollo, permitiendo una evaluación temprana y admitiendo modificaciones que tengan un efecto positivo en cuanto a costo, tiempo y satisfacción del cliente.

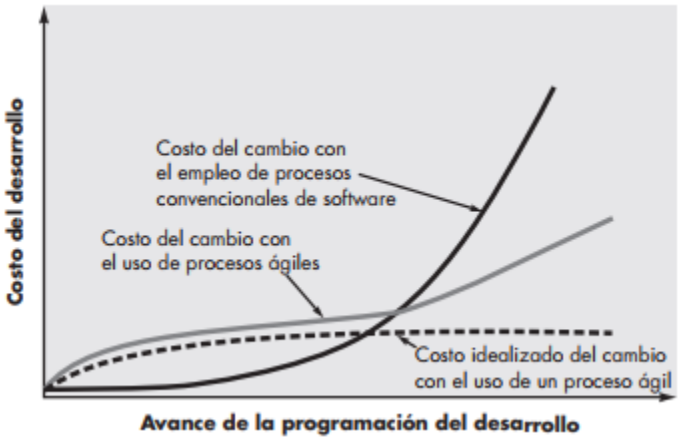


Figura 6: Comparativo de costos del cambio entre metodología tradicional vs metodología ágil
Fuente: Ingeniería del Software – Un enfoque práctico.

2.3.1 PRUEBAS CASCADA

Este proceso es el que se conoce de manera ortodoxa o tradicional, donde, al terminar la etapa de desarrollo, pasa al área de QA para iniciar el proceso de pruebas. Posterior a esto, viene el proceso de estabilización o *bug fixing*, cuyo objetivo es reparar los defectos encontrados [5].

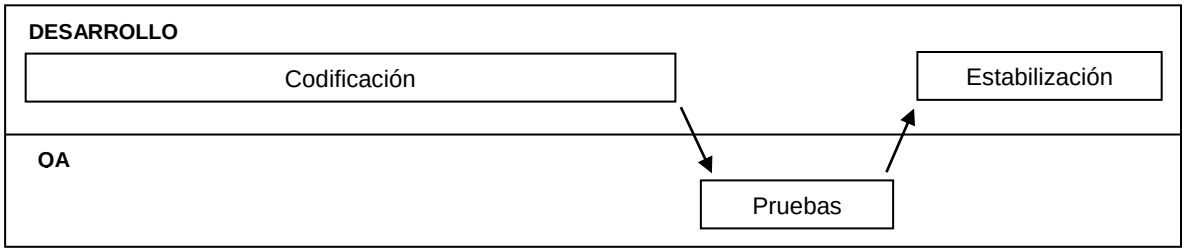


Figura 7: Testing tradicional
Fuente: Elaboración Propia

Algunos de los riesgos asociados a este tipo de estrategia son:

1. Al ser un proceso lineal y de tiempos bien establecidos, si la etapa de desarrollo toma más de lo planificado, se reduce el tiempo para la etapa de pruebas. Asimismo, puede ocurrir para la etapa de estabilización cuando el proceso de pruebas toma más del tiempo estimado.

2. Existen tiempos muertos para el área de desarrollo cuando se está en la etapa de pruebas, o para el área de QA cuando se produce la estabilización por parte de desarrollo.

3. La falta de integración entre desarrollo y QA al funcionar como áreas independientes desfavorece el trabajo en equipo.

2.3.2 PRUEBAS MODELO EN V

El modelo en V es una variante del modelo en cascada que integra pruebas en cada etapa del desarrollo, donde en cada nivel, el *tester* debe asegurar que los resultados cumplen con la validación y verificación del *software*. Mediante esto, se pretende establecer la etapa en específica a la cual se debe regresar en caso de encontrar algún defecto.

Cabe hacer la diferencia entre validación y verificación. La primera hace referencia a las actividades enfocadas a asegurar que el sistema respeta los requerimientos del cliente, es decir, si se está construyendo el producto correcto. Mientras que verificación habla sobre el proceso de asegurar la implementación correcta de una funcionalidad, es decir, si se está construyendo el producto correctamente [28].

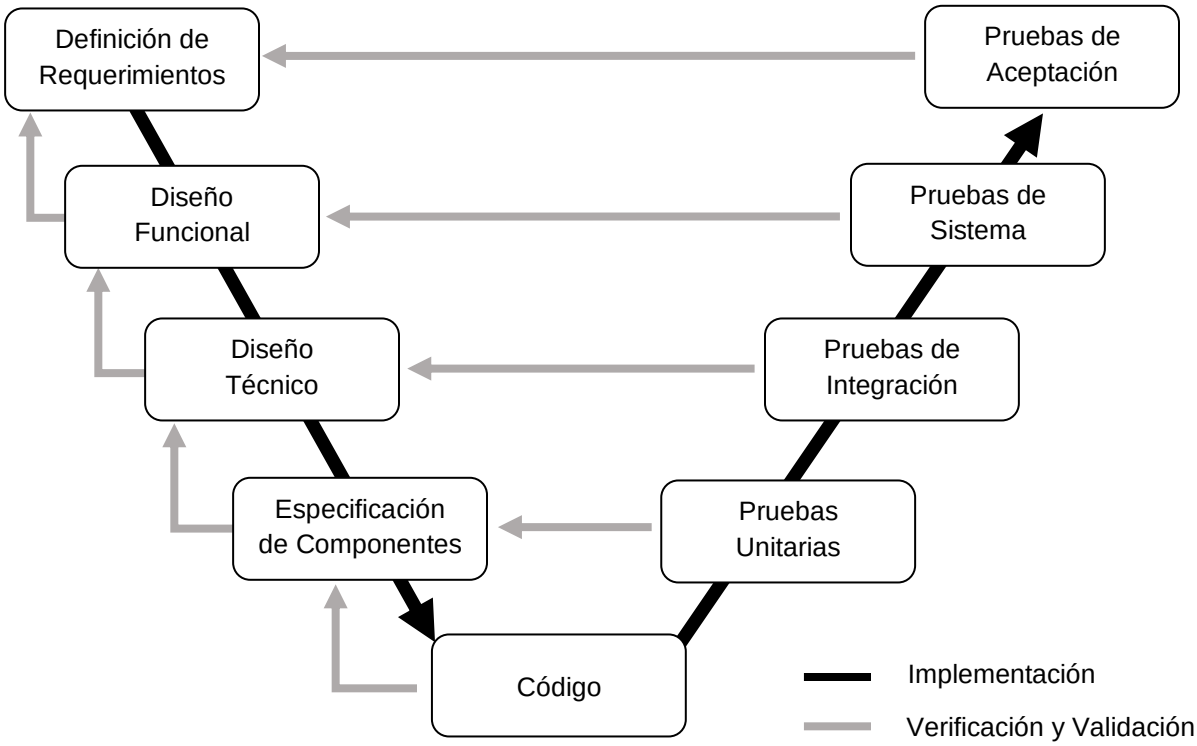


Figura 8: Modelo en V
Fuente: Elaboración Propia

2.3.3 CROSS TESTING

Esta manera de ejecutar pruebas trabaja con base en la revisión cruzada del código, en donde un desarrollador revisa o prueba el trabajo realizado por el otro. En esta forma de trabajo, el equipo de QA está dentro del área de desarrollo lo que le entrega agilidad a la hora de entregar un producto final [7].

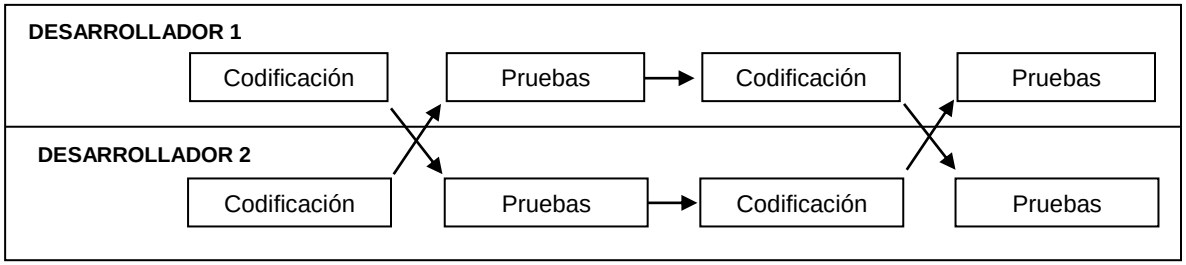


Figura 9: Cross testing
Fuente: Elaboración Propia

2.3.3 CICLO DE 2 SPRINT (ESTILO CASCADA)

Esto está ligado a lo que es metodología ágil, que se basa en la liberación de entregables del código en cada ciclo o iteración que vaya a tener el proyecto. Cada una de estas iteraciones es a lo que se le conoce como Sprint. Durante el primer Sprint, el equipo de desarrollo realiza la codificación mientras el equipo de QA trabaja en la matriz de casos de prueba.

En el segundo Sprint, el equipo QA realiza las pruebas del código creado del primer Sprint (entregable) y crea la matriz de casos de prueba que se utilizará para probar el siguiente entregable. Por su parte, el equipo de desarrollo realiza la estabilización del primer código y crea el correspondiente al segundo Sprint [7].

Si bien es una forma que elimina los tiempos muertos en gran medida, no es muy aconsejable debido a que el objetivo de cada Sprint es entregar una porción de código probada y funcionando [4].

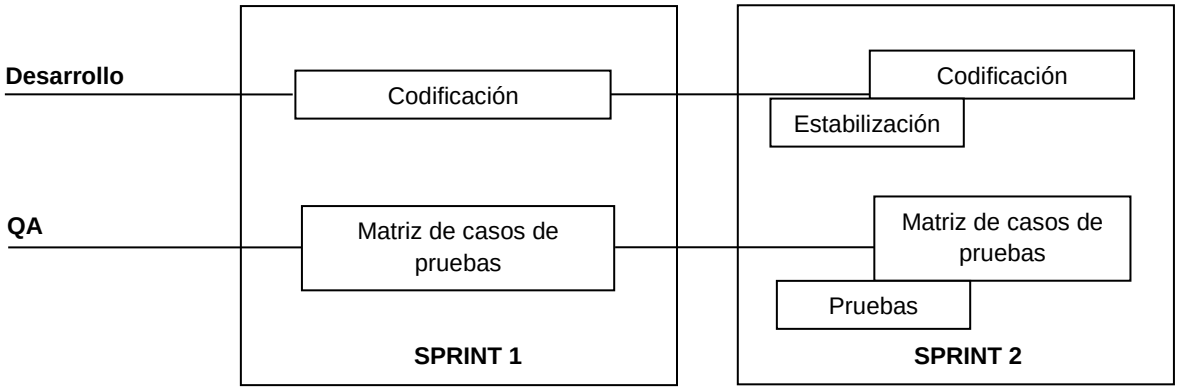


Figura 10: Ciclo de 2 Sprint
Fuente: Elaboración Propia

2.3.4 UN SPRINT

Este estilo también se enmarca dentro del marco de lo que son las metodologías ágiles y su proceso es similar a lo que son las pruebas dentro de un estilo tradicional, esto es, se inicia con el proceso de codificación de los requerimientos, posteriormente se realiza el proceso de pruebas por parte del equipo de QA basándose en una matriz de casos de prueba previamente generada con base en los requisitos y finalmente se continua con el proceso de estabilización, donde se solucionan los defectos encontrados por el área de QA. Lo que lo diferencia del estilo tradicional, es que todo este proceso se realiza dentro de un Sprint [7].

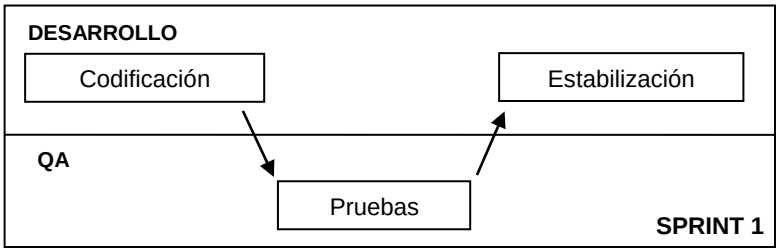


Figura 11: Un Sprint
Fuente: Elaboración Propia

Presenta las mismas falencias que la estrategia basada en gestión tradicional, sin embargo, para mitigar el problema de los tiempos muertos se pueden incorporar las tareas realizadas en el ciclo de dos Sprint: realización de casos de pruebas mientras se codifica y codificación de funcionalidades del siguiente Sprint mientras se ejecutan las pruebas por parte de QA [7].

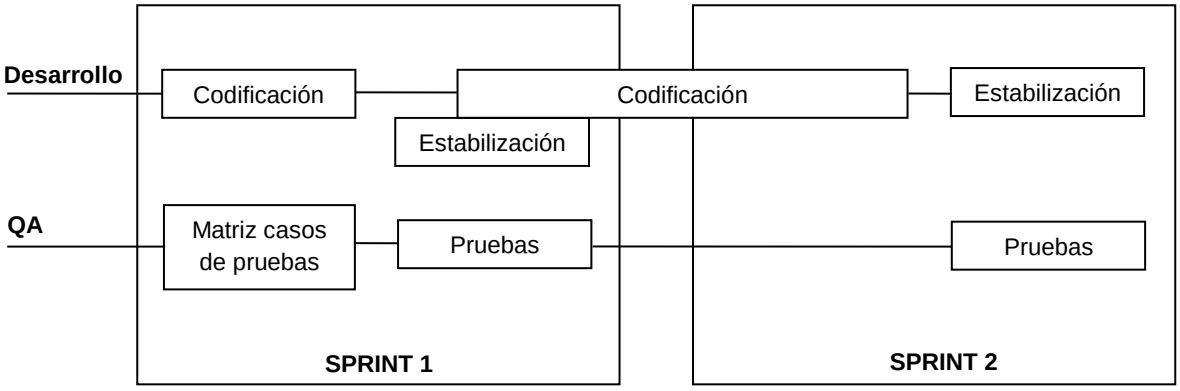


Figura 12: Un Sprint optimizado
Fuente: Elaboración Propia

2.3.5 ENFOQUE ÁGIL

Su principal objetivo es incorporar de forma más rápida al área de QA en el proceso de pruebas tomando funcionalidades más pequeñas de código. Esto requiere un buen análisis previo del aplicativo en su totalidad, lo que le otorga a este estilo cierta complejidad [7].

Los primeros días, cuando el equipo de desarrollo está en proceso de creación de la primera pieza de software, no existe nada que revisar, lo que genera tiempos muertos para el equipo de QA.

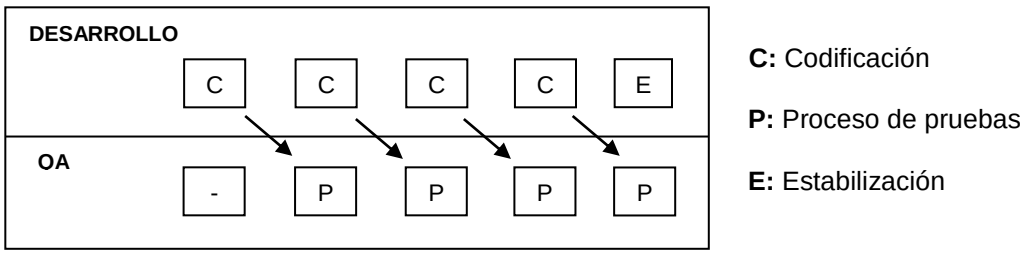


Figura 13: Enfoque ágil
Fuente: Elaboración Propia

2.3.6 CARRY OVER

Se genera con el fin de mejorar los tiempos de ambos roles y maximizar el trabajo dentro de un Sprint. Consiste en pasar al siguiente Sprint aquellos requerimientos que no se alcanzan a finalizar, realizando una reestimación de tiempos al incorporar nuevas tareas al ciclo siguiente [7].

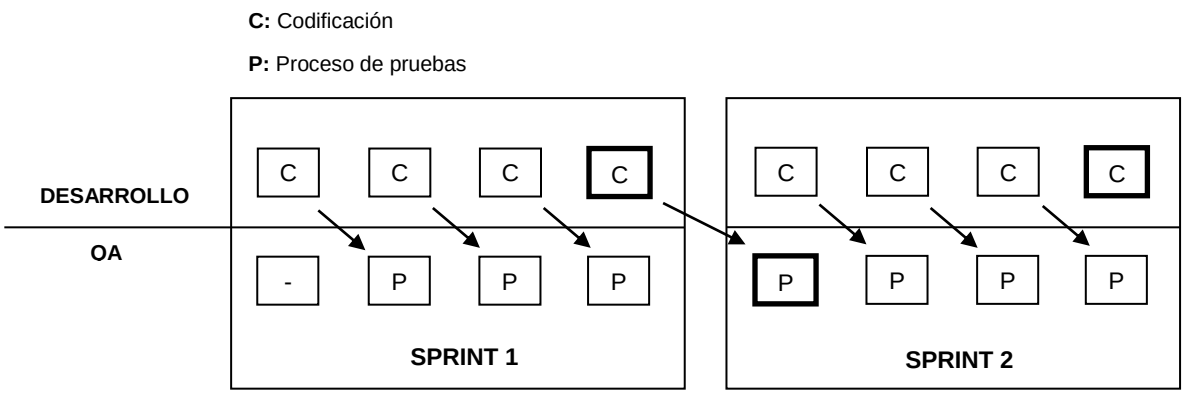


Figura 14: Carry Over
Fuente: Elaboración Propia

2.4 PRUEBAS MANUALES

Las pruebas manuales son aquellas que son ejecutadas directamente por personal capacitado del área de QA, cuyo objetivo es simular las acciones que realiza un usuario en el aplicativo una vez sea liberado para su uso. Se basan en una matriz de casos de prueba que cubre la totalidad o parte de las funcionalidades del aplicativo (obedeciendo a la metodología utilizada) que, dependiendo del resultado que se obtenga, permite tomar decisiones respecto a la calidad del desarrollo, modificación de alcances o, en definitiva, la liberación del aplicativo como producto terminado.

El proceso de generación de una matriz de cobertura considera los siguientes hitos:

1. **Levantamiento de requerimientos:** Proceso mediante el cual, el dueño del producto define con los involucrados las funcionalidades que requiere y el objetivo que persigue. Cabe destacar que este no es un evento único, se itera tantas veces sea necesario hasta que todos tengan el mismo entendimiento de los objetivos planteados.

2. **Documentación:** Una vez terminado el proceso de levantamiento, se procede a documentar los acuerdos tomados en un Documento de Requerimientos, en donde se detallan los casos de uso y se describen las funcionalidades a desarrollar en cuanto a alcance y comportamiento.

3. **Generación de matriz de cobertura:** Es el último hito, en donde se crean los casos de prueba que validarán que las funcionalidades cumplen las expectativas del dueño del producto. Se debe señalar que el Documento de Requerimientos no

necesariamente entrega toda la información que se requiere para generar los casos de prueba, por lo que muchas veces es necesaria su discusión con el equipo de desarrollo y el usuario final, quienes entregarán información como paso a paso y datos de usuario.

Ventajas:

- Al ser realizadas por un humano, se realiza un análisis más profundo de la funcionalidad que se está probando, lo que contribuye a mejorar la experiencia de un usuario (pruebas de usabilidad).
- Entrega mayor dinamismo en cuanto a la detección de defectos y a la mejora de la calidad, debido a que se pueden realizar pruebas que no estén contempladas dentro de la matriz de casos a medida que el *tester* se encuentra en el proceso de ejecución.
- No se requiere de herramientas complejas para el manejo de casos de pruebas, evaluación de resultados, administración de evidencias y reporte de defectos al área de desarrollo. Si bien el uso de una herramienta especializada ayuda a la gestión como Testlink, también es posible utilizar herramientas de ofimática durante el proceso: matrices realizadas en Excel y registro de evidencias en Word.
- Le otorga cierto grado de agilidad al proceso de resolución de defectos, debido que se puede reportar de manera inmediata al equipo de desarrollo mientras el *tester* continúa con el proceso de certificación.

Desventajas:

- La acción repetitiva de pruebas se vuelve agotadora y tediosa, lo que podría generar que al *tester* se le pasen defectos.
- En un modelo tradicional, los tiempos de duración de los ciclos de pruebas pueden llegar a ser muy extensos, lo que impacta en la planificación del proyecto.
- El avanzar más rápido en un proceso de pruebas manuales, implica necesariamente la incorporación de más recursos que apoyen a la certificación, lo que aumenta los costos del proyecto.

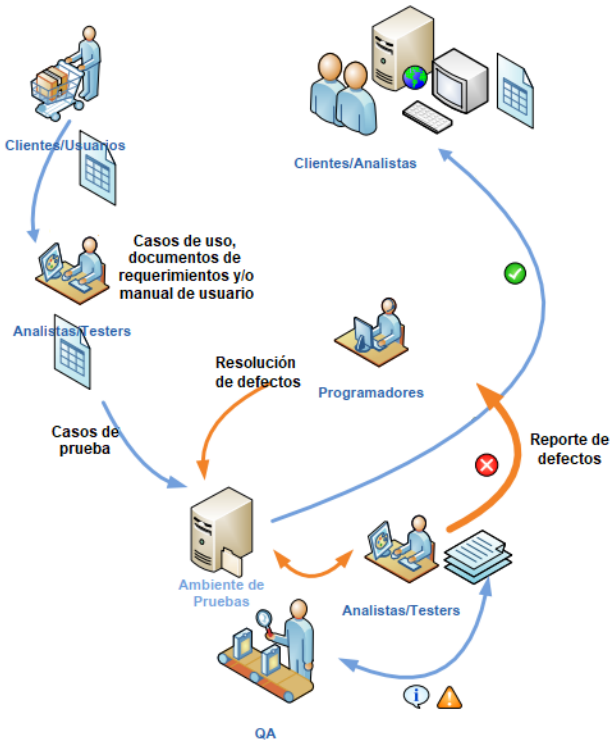


Figura 15: Proceso de ejecución de casos de pruebas manuales
Fuente: it-Mentor

Las labores que debe ejecutar un *tester* cada vez que inicia un proceso de certificación como QA manual es el siguiente:

1. Ejecutar las pruebas consideradas para el proceso de certificación de acuerdo al detalle y paso a paso descritos en la prueba.
2. Registrar el estado de cada prueba, esto es, si es un caso que termino correctamente o presento algún defecto.
3. Registrar las evidencias (*screenshots*) de cada paso realizado en cada prueba.
4. Una vez finalizado todas las pruebas establecidas, se realiza un informe de finalización en donde se detalla el estado final de la certificación (cantidad de estados OK, NO OK y/o Bloqueados)

2.5 PRUEBAS AUTOMATIZADAS

Consiste en el proceso de ejecución de pruebas sin la intervención humana, en donde, mediante un script de programación que emula la acción de un *tester* manual, se compara el resultado obtenido con el esperado [7].

El decidir crear pruebas automáticas implica la utilización de otros recursos, como herramientas de programación y su eventual compra de licencias, equipos con determinadas características y personal especializado, lo que repercute directamente en los costos asociados a un proyecto, es por esto que es necesario saber qué se debe automatizar. Para esto, Mike Cohn y Anand Bagmar proponen lo que se conoce como "pirámide de pruebas", donde sugieren en qué grado se deberían automatizar las pruebas de software [21].

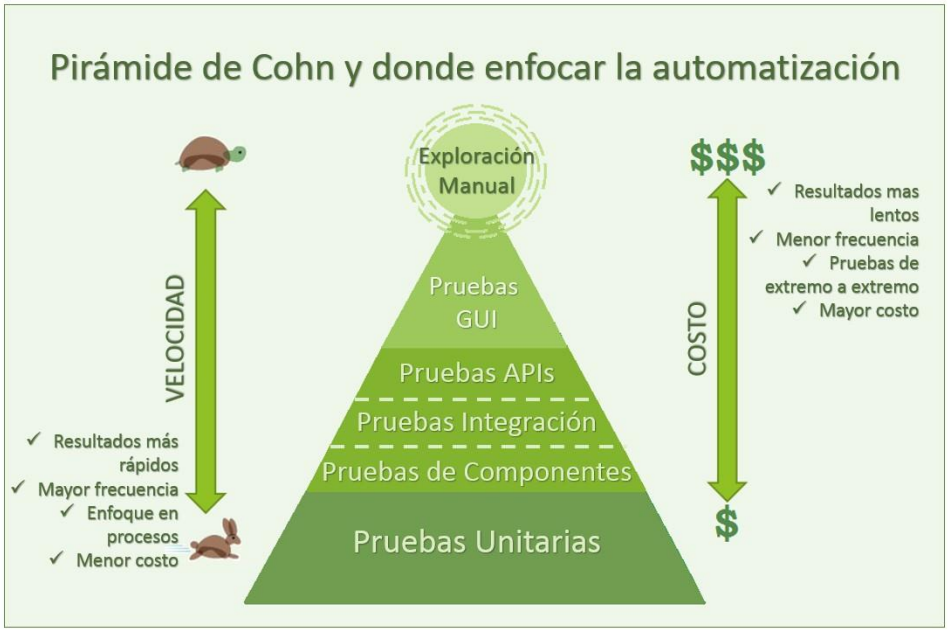


Figura 16: Pirámide de pruebas
Fuente: Elaboración propia

Principalmente, se propone crear gran cantidad de test unitarios, puesto que con esto se podrían detectar fallos de forma rápida y en la fase de desarrollo, donde el costo de reparar defectos es menor y es fácil prever que si algo falla en estas pruebas, es muy probable que las pruebas de integración también fallen. Las pruebas de UI debiesen ser las menos, debido a que corresponden a interfaces más bien gráficas, las que son difíciles de automatizar y lentas en el proceso de ejecución por posibles dependencias con otros componentes [7].

Ventajas:

- Las pruebas ejecutan siempre las mismas operaciones, lo que elimina el error humano.
- Permite ejecutar muchas iteraciones de las mismas pruebas, lo que ayuda a una estabilización temprana del código.
- El proceso de certificación puede ser ejecutado de forma desatendida en horarios no laborales.

Desventajas:

- No es posible medir la usabilidad de un aplicativo con las herramientas de automatización.
- Se requiere conocimiento especializado de programación para adaptar los scripts automatizados a los requerimientos.
- Los scripts pueden requerir mantenimiento.

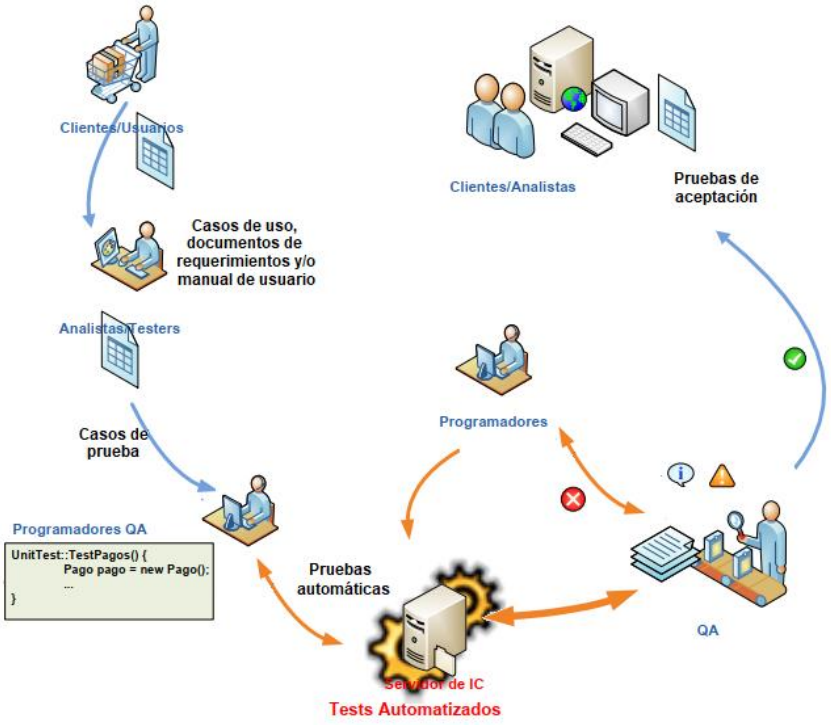


Figura 17: Proceso de ejecución de casos de pruebas automáticas
Fuente: it-Mentor

2.5.1 TECNICAS DE AUTOMATIZACIÓN

Existen múltiples técnicas de programación de pruebas que tienen como finalidad facilitar la generación de *scripts*, minimizar los tiempos y apoyar al desarrollo para reducir al mínimo los errores. Dentro de estos, los que más destacan actualmente son [7]:

a) *Test Driven Development* (TDD)

Es una técnica que surge con XP en la cual se busca tener un código sin errores. Se parte por la construcción del producto con las pruebas que fallan debido a que la funcionalidad aún no está construida. Luego se crea el código de la forma más simple posible, de tal forma que la prueba se ejecute sin errores. Se continúa con la refactorización del código y finalmente se va iterando hasta tener la funcionalidad terminada.

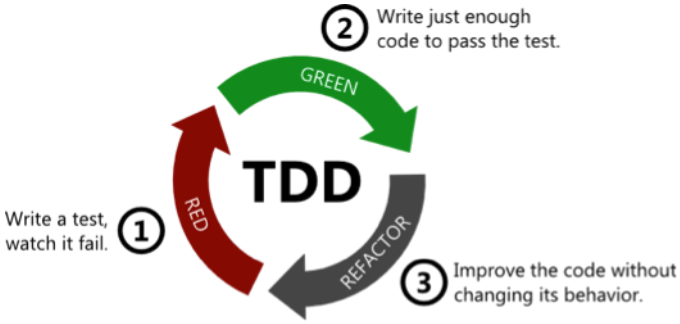


Figura 18: Ciclo TDD
Fuente: https://www.techwell.com

El proceso costa de 3 etapas:

1. ***RED*:** Su finalidad es generar las pruebas unitarias con anterioridad al código.
    - Sirven para verificar el buen funcionamiento de un módulo
    - Se deben ejecutar sin intervención manual.
    - Repetibles cuantas veces sea necesario.
    - Su finalidad es cubrir la totalidad del código.
    - Deben ejecutarse de forma correcta en cualquier ambiente.
    - Son independientes las pruebas entre sí.

2. ***GREEN*:** Se implementa el código que hace pasar la prueba creada para el módulo.
    - Implementación básica del código solo para hacer pasar la prueba.
3. ***REFACTOR*:** Refactorizar o mejorar el código, con el objetivo de eliminar la deuda técnica[1].
    - Se modifica el código del software sin modificar su comportamiento.
    - No se utiliza para agregar funcionalidades.

Ventajas:

- El equipo se enfoca en lo que el cliente necesita.
- Al tener pruebas tempranas, se tendrá mayor calidad en el proceso de construcción de software
- Al existir las pruebas antes del desarrollo, el objetivo se alinea con los requerimientos de usuario.

Desventajas:

- Se requiere un mayor conocimiento técnico.
- Se aplica mayoritariamente a capas de negocio y acceso a datos.
- Implica mayores tiempos al proyecto al tener que ejecutar las pruebas cada vez que se crea una pieza de código.

b) *Acceptance Test Driven Development* (ATDD)

Es el proceso en que todo el equipo revisa, de manera conjunta, los criterios de aceptación[2] de los requerimientos. El flujo comienza con una reunión entre los involucrados, en donde se crean los distintos escenarios posibles que puede tener como comportamiento un requerimiento para luego construir una prueba automatizada por cada uno de estos escenarios.

[1] Se refiere a los problemas que un *software* puede tener debido a la falta de prolijidad o buenas prácticas durante su construcción.
[2] Condiciones que un requerimiento debe tener para ser aceptado por un usuario.

c) *Behavior Driven Development* (BDD)

Se puede considerar como una extensión de TDD, debido a que su flujo es similar, pero el alcance es distinto: mientras en TDD se busca tener un código lo más pulcro posible, en BDD se busca que se cumpla el requerimiento de usuario introduciendo un lenguaje específico que no necesita conocimiento técnico denominado Gherkins. Este permite que el cliente escriba las pruebas desde su punto de vista y guiar la generación del código enfocada en el comportamiento.

La generación de escenarios con Gherkins se realiza bajo una estructura determinada:

***Feature*:** Funcionalidad que se desea revisar.

Scenario: Descripción del escenario que será revisado.

Given: Contexto del escenario el cual define el comportamiento esperado.

When: Acciones que se realizarán en la ejecución de la prueba.

Then: Resultado esperado.

Existe una gran variedad de herramientas que permiten crear y ejecutar pruebas automatizadas, por lo que se debe definir claramente cuál es el alcance que se quiere lograr, el nivel de cobertura y el aplicativo que será sujeto de automatización, puesto que esto permitirá decantar por una herramienta u otra. A continuación, se presenta un cuadro comparativo entre las diferentes herramientas disponibles que, si bien no son todas, permite tener una idea de la cobertura que se puede tener en la actualidad con las diversas opciones que existen.

Como se aprecia en la Tabla 3, las aplicaciones de pago son las que, en general, poseen mayor cantidad de funcionalidades integradas en un mismo software, facilitando su uso en distintos ambientes. Las aplicaciones gratuitas pueden incluir otras opciones de uso, pero deben ser complementadas con otras herramientas que permitan su utilización. En la tabla 4, se presenta un cuadro comparativo que permite integrar la automatización *mobile* con las aplicaciones de software libre.

Tabla 3: Cuadro comparativo herramientas de automatización.
Fuente: Elaboración Propia

	SELENIUM	KATALON STUDIO	UFT (UNIFIED FUNCTIONAL TESTING)	TESTCOMPLETE	SOAPUI
DISPONIBLE	2004	2015	1998	1999	2005
APLICACIÓN	• Web	• Web	• Web • Mobile Apps • Escritorio	• Web • Mobile Apps • Escritorio	• Web Services
COSTO	Libre	Libre	Pago	Pago	Pago
PLATAFORMA SOPORTADA	• Window • Linux • OS X	• Window • Linux • OS X	• Window	• Window	• Window • Linux • OS X
LENGUAJE SOPORTADO	• Java • Javascript • Python • Ruby • Perl • PHP • C#	• Java • Groovy	• VBScript	• Javascript • Python • VBScript • Delphi • Jscript	• Groovy
HABILIDADES DE PROGRAMACIÓN	Compleja integración a diferentes frameworks	No requeridas. Especial para pruebas avanzadas	No requeridas. Especial para pruebas avanzadas	No requeridas. Especial para pruebas avanzadas	No requeridas. Especial para pruebas avanzadas
FACILIDAD DE USO	• Requiere conocimientos avanzados para su uso y aplicación	• Fácil de configurar y usar	• Compleja instalación • Requiere capacitación para su uso	• Fácil de configurar • Necesita capacitación para su uso	• Fácil de configurar y usar

Tabla 4: Cuadro comparativo herramientas de automatización *mobile*.
Fuente: Elaboración Propia

	ROBOTIUM	CALABASH	UIAUTOMATOR	APPIUM	ESPRESSO
ANDROID	SI	SI	SI	SI	SI
IOS	NO	SI	NO	SI	NO
LENGUAJE SOPORTADO	• Java	• Ruby	• Java	• Java • Javascript • Python • Ruby	• Java
HERRAMIENTA CRACIÓN DE SCRIPT	Testdroid Recorder	CLI	UI Automator Viewer	-	Hierarchy Viewer

La cantidad de casos a automatizar es directamente proporcional a la envergadura del proyecto, por lo que, para proyectos grandes, se hace imprescindible llevar un correcto control de las automatizaciones para evitar utilizar tiempo innecesario en la identificación de aquellos casos que son útiles para una determinada certificación o de los que requieren refactorización cuando se modifica un componente del aplicativo. Para evitar estos inconvenientes, es que existen herramientas que permiten llevar un orden y control a nivel de identificación, versionamiento y/o ejecución, con lo que se mantiene un correcto control del conocimiento, cuando de automatizaciones se trata, a través de su información histórica y las descripciones que permiten incorporar a cada caso.

A modo de resumen, se presenta un cuadro comparativo de las disponibles actualmente en el mercado y los alcances de cada una:

Tabla 5: Cuadro comparativo herramientas de gestión de casos de prueba
Fuente: Elaboración Propia

		Costo	Gestión				Defectos		Reportes		
		Licencia	Plan de pruebas	Caso de prueba	Trazabilidad de casos	Ejecución	Reporte	Historial de defectos	Informe	Dashboard	Evidencia*
TesLodge		Pago	✓	✓	✗	✓	✓	✓	✓	✗	✓
TcLab	TcLab	Pago	✓	✓	✓	✓	✗	✗	✗	✗	✓
qTest		Pago	✓	✓	✓	✓	✗	✗	✓	✓	✓
PractiTest	PractiTest	Pago	✓	✓	✗	✓	✗	✗	✓	✓	✓
Zephyr	ZEPHYR	Pago	✓	✓	✓	✓	✓	✓	✓	✓	✓
Testcollab	Testcollab	Gratis (limitado)	✓	✓	✓	✓	✓	✓	✓	✓	✓
TestLink	TestLink	Gratis	✓	✓	✗	✓	✓	✗	✓	✗	✓
MicroFocus	MICRO FOCUS	Pago	✓	✓	✗	✓	✓	✗	✓	✓	✓
TestRail	TestRail	Pago	✓	✓	✗	✓	✓	✗	✓	✗	✗
Testuff	testuff	Pago	✗	✗	✓	✓	✓	✗	✓	✗	✓

Para la clasificación de la columna de Evidencias, se considera integración con otras herramientas de reporte de defectos, lo que permite el registro y manejo de las evidencias de las ejecuciones, ya sea manuales o automatizadas.

CAPÍTULO 3: PROPUESTA DE SOLUCION

El modelo que se propone se basa en 4 acciones que deben ser gestionadas por lo involucrados en la organización; éstas son: Identificación, Análisis y Diseño, Ejecución, Control. Este modelo supone una relación unidireccional entre una tarea y otra, las cuales deben ser ejecutadas de manera secuencial y ordenada. El resultado obtenido en cada una será la entrada para la tarea siguiente, por lo que su orden es estricto e inamovible. En caso de que una de estas tareas otorgue entradas complejas de evaluar, es necesario hacer un *rollback* y replantear los objetivos propuestos.

3.1 IDENTIFICACIÓN

Es la etapa relevante del modelo que tiene por objetivo determinar si las condiciones son las óptimas para realizar la automatización. Una fase clave que, de no ser considerada, podría ocasionar riesgos difíciles de controlar en el proyecto.

Como antecedente, es importante tener claro que, el decidir realizar pruebas automáticas necesariamente implica la utilización de otros recursos como herramientas de programación y su eventual compra de licencias, equipos con determinadas características y personal especializado, lo que afecta directamente los costos asociados a un proyecto, es por esto que es de suma importancia identificar cuándo y qué automatizar.

El precedente para la automatización son los casos manuales, los cuales corresponden a la base de la generación de los *scripts*, siendo necesarios para obtener los flujos de ejecución, precondiciones y resultados esperados de las pruebas. Es por esto que, como paso previo, se debe realizar un análisis de la matriz de cobertura, persiguiendo como objetivo la exclusión de aquellos casos redundantes, la verificación de la claridad del detalle del caso y la eliminación de ambigüedades en la interpretación del resultado.

Una vez analizada la cobertura, se debe identificar si se está en una situación óptima para realizar pruebas automatizadas dentro de un proyecto o proceso. Para esto se propone la evaluación de los siguientes elementos clave (similar a lo que plantea McCall en su modelo):

Tabla 6: Tabla de ponderación.
Fuente: Elaboración Propia

Índice	Parámetro	Evaluación	Detalle	Ponderación
1	**Criticidad**	1 o 0	1 si más del 50% de las pruebas son críticas para el flujo del negocio y 0 para no	0.3
2	**Portabilidad**	1 o 0	1 si las pruebas se realizan en más de un navegador o plataforma y 0 para no	0.15
3	**Perfilamiento**	1 o 0	1 si las pruebas deben ser realizadas por distintos perfiles de usuario y 0 para no	0.2
4	**Frecuencia**	1 o 0	1 si las pruebas se repiten por cada nueva certificación y 0 para no	0.25
5	**Complejidad**	1 o 0	1 si más del 9% de las pruebas que son complejas (en cuanto a tiempo invertido en realizarlas de forma manual y dificultad de reproducción) y 0 para no.	0.1

Luego de haber evaluado cada uno de estos 5 puntos, se debe obtener un resultado en base a la ponderación de la siguiente manera:

$$\sum_{1}^{5} Evaluación_n * Ponderación_n$$

Figura 19: Fórmula de ponderación de criterios
Fuente: Elaboración propia

La interpretación de este resultado se basa en la siguiente tabla:

Tabla 7: Interpretación de tabla de ponderación
Fuente: Elaboración Propia

Resultado	Descripción
0 - 0.45	No apto para automatización
0.5	Posible automatización. En el corto plazo no aportaría una mejora sustancial al proceso, por lo que queda a decisión de los involucrados donde la decisión se debe basar en disponibilidad de recursos humanos y económicos.
0.55 - 1	Automatizable

Si la evaluación indica que se está en la situación ideal, es necesario identificar qué automatizar, para lo cual se debe utilizar el diagrama de flujo de la Figura 20 que entrega resultados mediante un proceso en 3 niveles: proyecto, matriz de cobertura y caso de prueba.

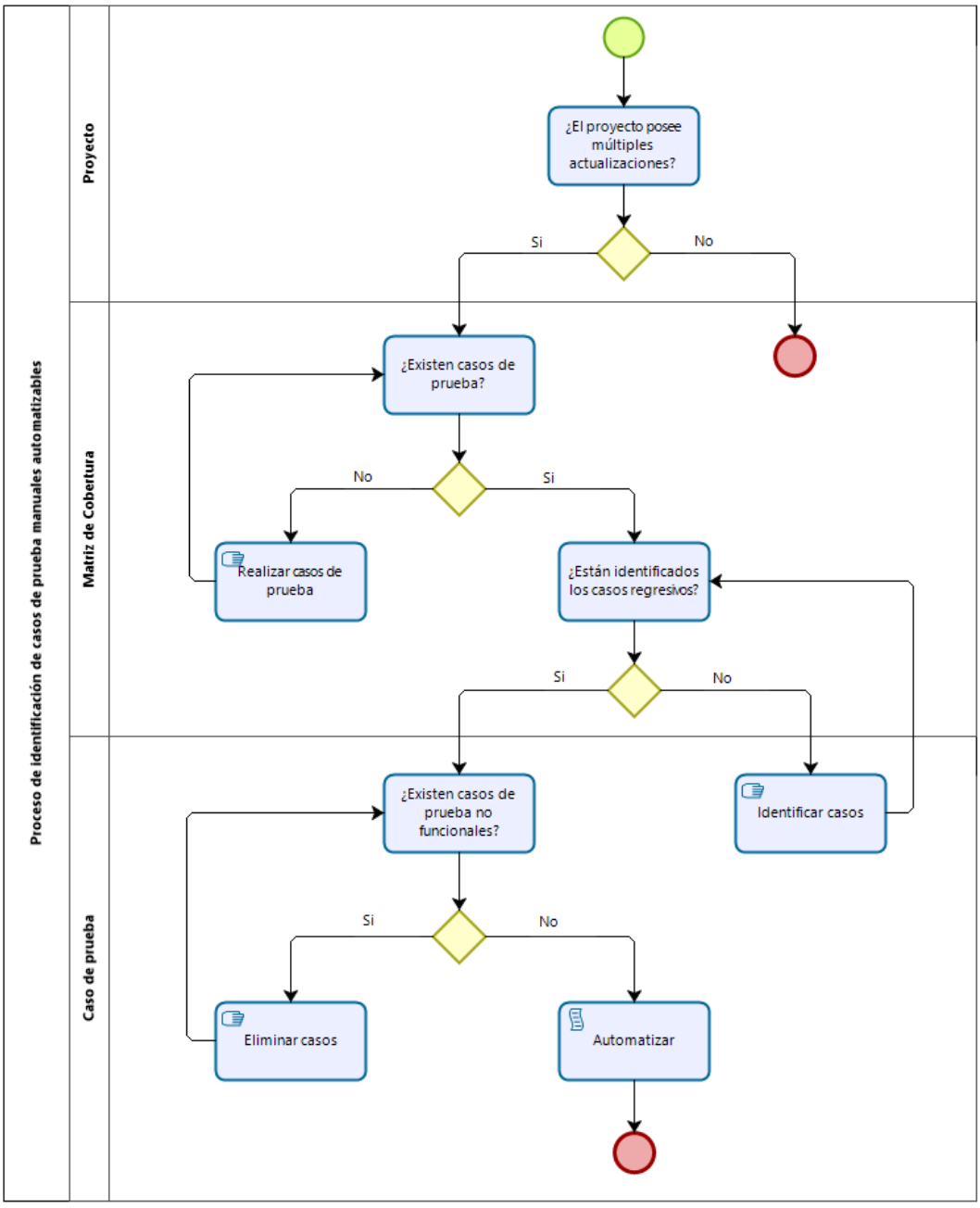


Figura 20: Proceso de identificación de pruebas automáticas
Fuente: Elaboración propia

Este flujo da una buena aproximación de aquellos casos que deben ser considerados a la hora de la automatización, sin embargo, existen casos relevantes para el aplicativo que pudieran quedar fuera, por lo que se deben responder las siguientes preguntas para afinar el criterio y obtener el mejor alcance posible:

- ¿Cuáles son los casos de prueba que se tendrán que ejecutar una mayor cantidad de veces?

  La repetitividad de las pruebas es un factor alto a la hora de tener en cuenta, pues finalmente se consideran regresivos y que evalúan la estabilidad del sistema.

- ¿Cuáles son los casos o procesos funcionales del sistema que deben ser ejecutados sin errores?

  Permite identificar aquellos casos que son críticos para el sistema y para los cuales es preciso eliminar el error humano generado por el agotamiento en la ejecución de pruebas repetitivas, costosas, tediosas o complejas.

- ¿Existen pruebas recurrentes que demoran tiempo excesivo en ser ejecutados por un *tester*?

  Si la respuesta es “sí”, es buena candidata a automatización.

- ¿Se realizarán pruebas no funcionales?

  Acá se consideran pruebas de estrés, carga, desempeño, concurrencia, etc., las cuales sí se deben automatizar.

- ¿Es muy costoso automatizar la prueba?

  Esto debe ser respondido por un experto y que va a filtrar si todas las respuestas anteriores arrojaron resultados afirmativos. Este basará su respuesta en la complejidad y tiempo que tome la construcción de la prueba, lo cual repercute directamente en los costos económicos. Si la respuesta es “sí”, no se debe incorporar dentro del plan.

3.2 ANALISIS Y DISEÑO

Cuando se automatizan pruebas, el especialista debe hacerlo teniendo una visión de futuro, esto es, ponerse en la situación que cada *script* será usado continuamente, cada vez que se requiera mientras el sistema para el cual fue desarrollado sufra modificaciones.

Es común que, a medida que pasa el tiempo, las nuevas tecnologías impulsen nuevos requisitos en el aplicativo que pueden cambiar completamente su estructura, por lo que uno de los objetivos principales a alcanzar al momento de comenzar con la codificación de las pruebas es la fácil mantención. Para obtener esto, es importante tener un entendimiento acabado del aplicativo en su totalidad, esto es, funcionalidades, subfuncionalidades y reglas de negocio consideradas en el desarrollo, por lo que se debe contar con dos documentos principales: Manual de Usuario y Documento de Requerimientos. Con estos, es posible evaluar cómo se procede con cada una de las opciones para ejecutar una acción e identificar aquellas que tienen comportamiento común, las que, posteriormente, se convertirán en *scripts* Genéricos. Estos Genéricos son similares a las Funciones en el mundo del desarrollo de *software*, con la diferencia que su objetivo principal es ejecutar una acción dentro de un aplicativo. Independiente de la herramienta que se decida ocupar para automatizar, es importante que aquellos Genéricos que tengan un comportamiento similar sean desarrollados en una misma pieza de código, de tal forma que faciliten su identificación y posterior llamado, además que ayuda a los especialistas a realizar un mantenimiento más eficiente.

Se debe tomar en cuenta que, cuando los casos de prueba manuales son creados, su finalidad no es que se conviertan en automatizaciones a futuro, sino que alcancen la mayor cobertura posible, por lo que son susceptibles de análisis una vez ejecutada la etapa de Identificación, debido a que es muy común que éstos sean demasiado grandes para ser automatizados, por lo que deben ser descompuestos cuando sea necesario para permitir que el proceso de creación de pruebas sea más rápido y eficaz. A esto se le llama **Modularización de Casos** y al momento de entrar en esta fase es bueno considerar cada respuesta del sistema como un caso distinto.

Cuando se comienza con la programación de las pruebas, se debe evaluar el avance diario teniendo como métrica que, lo normal, es crear entre 3 y 5 casos diarios. Si el especialista identifica que está por debajo de esta métrica, es probable que la etapa de modularización no fue realizada de la forma correcta, por lo que se debe detener la codificación, volver a analizar los casos y generar una nueva propuesta de automatización.

La automatización, como un proyecto de desarrollo más, no está exenta de buenas prácticas que deben ser consideradas desde el inicio de la propuesta de solución, debido a que esto servirá de base para generar una estimación de tiempos adecuada y acorde a las necesidades del proyecto, alineando alcances y aterrizando expectativas de usuario:

1. **Comentarios:** Los comentarios ayudan a entender el código de una manera más rápida para aquellos que ven el código por primera vez, por lo que éstos deben ser concisos sin abusar de ellos, debiendo ser escritos solo en aquellas estructuras que impliquen un proceso complejo o involucren una regla de negocio específica.

2. **Parametrización:** Todo proyecto involucra la utilización de datos, ya sea para hacer *login,* para completar algún tipo de formulario, para ingresar a una determinada URL, etc., que pueden ser utilizados en distintas partes del aplicativo o que pueden ser cambiados cuando se actualice el sistema. Para facilitar esto es que se debe evitar la escritura "en duro" de esta información dentro del código y fomentar la parametrización de datos. El cómo se realice esta parametrización depende mucho de la herramienta utilizada, puesto que algunas proporcionan métodos internos de manejo de datos, como UFT, por ejemplo, que posee un DataTable o Panel de Datos como planilla tipo Excel dentro de cada *script* que permite su administración, y otras, como Selenium, necesariamente deben ser complementadas con manejo de archivos para proporcionar dicha información.

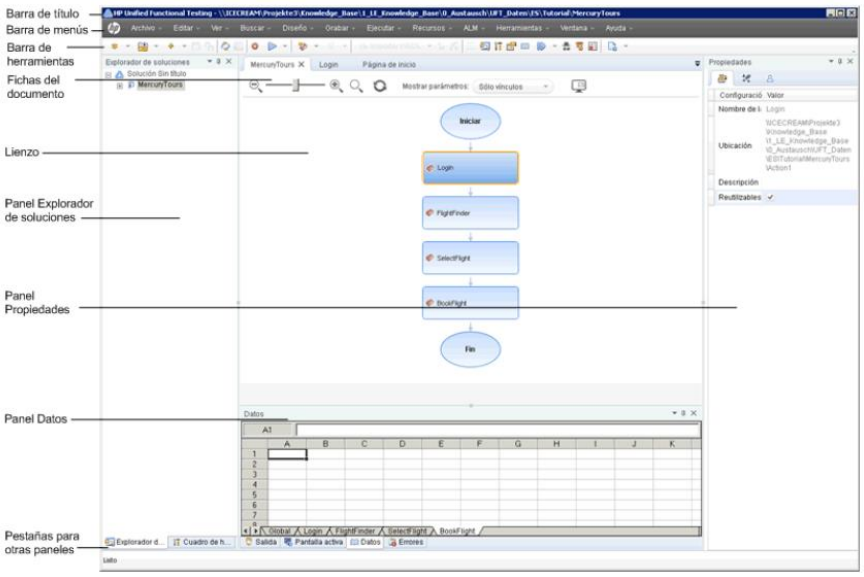


Figura 21: Vista de UFT con Panel de Datos
Fuente: Tutorial para pruebas UI – HP

3. **Control de errores:** Cuando se prueba un aplicativo, no solo se prueba el camino feliz, sino que también aquellos comportamientos inesperados que puede ejecutar un usuario que podrían ocasionar algún tipo de error en la ejecución, por lo que el especialista debe ser lo suficientemente capaz de ponerse en cada una de estas situaciones y realizar su control adecuado. Cada situación inesperada debe ser controlada con un mensaje entendible para el usuario que lo ejecuta, con el objetivo de que se puedan tomar decisiones con la información que esto entrega.

4. **Independencia:** Los casos de prueba manuales se crean de forma secuencial, de tal forma que el *tester* pueda continuar con la siguiente prueba sin tener que volver a comenzar desde el principio (*login*), sin embargo, esto no es posible con las pruebas automatizadas debido a dos inconvenientes principales: existen herramientas que cada vez que termina una prueba cierran el navegador y al comenzar la siguiente levantan una nueva vista, por lo que es imposible comenzar desde el punto en que quedó la prueba anterior. Por otro lado, otras permiten dejar el navegador abierto, pero cada vez que inicia una prueba, levantarán una nueva pestaña. Si bien con esta última opción es posible continuar desde el punto que quedó la prueba anterior entregándole la URL exacta y manteniendo el *token* de sesión, cuando son numerosas pruebas, el navegador colapsará con pestañas, provocando un efecto negativo en el rendimiento. Por estos motivos es que las pruebas deben ser independientes una de la otra, comenzado todas por el *login* y finalizando con el *logout* del aplicativo, cerrando el navegador.

   También se debe considerar esta premisa cuando existan casos de prueba que posean como precondición otro caso anterior, teniendo que ejecutar obligatoriamente este último dentro del flujo del caso a evaluar independiente si este caso-precondición ya ha sido ejecutado con anterioridad. De esta forma, se asegura que las pruebas no entregarán errores relacionados con problemas de ejecución.

5. **Evidencia:** Las herramientas de creación y ejecución de casos automáticos tienen incorporado dentro de sus funcionalidades, el registro final del caso que indica si éste fue ejecutado de manera correcta o si falló, ya sea durante el proceso de ejecución y si no se obtuvo el resultado esperado de acuerdo a las validaciones que se hayan ingresado en el código. El motivo del fallo lo proporciona el programador tal como se menciona en el punto 3, sin embargo, la complejidad de algunos aplicativos hace insuficiente la información que entrega un mensaje de error, por lo que siempre debe estar acompañado de una evidencia gráfica, la cual entregará al usuario mayor detalle respecto a la causa del error y se podrá actuar con

prontitud en caso que sea un defecto que afecte un flujo de negocio crítico. La gestión de estos registros varía de acuerdo a la herramienta usada: se pueden adjuntar en algún reporte automático o almacenarlas de manera local para su posterior evaluación.

3.3 EJECUCIÓN

Las pruebas automáticas consumen recursos de un dispositivo, por lo que se debe contar con una estación de trabajo exclusiva para su ejecución (notebook, computador de escritorio o máquina virtual) la cual no debe ser utilizada o intervenida en el momento en que estas pruebas estén corriendo para no generar ruido durante el proceso. Asimismo, no se debe perder el foco de la automatización: liberar un aplicativo con la mínima cantidad de errores a producción, por lo que se debe contar, además, con un ambiente dedicado para ejecutarlas que debe ser diferente al ambiente de producción y al de desarrollo.

Cuando llega el momento de ejecutar las pruebas, se deben tener en cuentas las siguientes consideraciones:

- En un *set* de casos no superior a 200, es aceptable considerar hasta un 10% de falsos positivos, esto es, que el caso se presente como “fallido” cuando lo que se está probando realmente no lo está. Esto ocurre debido a que las constantes llamadas al navegador y la utilización de recursos de la estación de trabajo, puede ocasionar que los despliegues de las vistas de usuario tomen mayor tiempo de lo que se midió al momento de la creación y prueba, por lo que el código, cuando intente buscar un objeto (botón, enlace, caja de texto, combobox, etc.) no lo encuentre y falle.

- En un set de casos superior a 200, es aceptable hasta 20 casos que se reporten como falsos positivos.

- Cuando los falsos positivos superen los indicadores mencionados, es causa de análisis e incluso de refactorización de los casos.

Se debe indicar que estas métricas deben ser consideradas sólo cuando se ejecuta el set de pruebas por primera vez, puesto que se espera sean corregidas (dentro de lo posible) apenas sean identificadas.

3.4 CONTROL

Los casos de prueba automáticos perduran en el tiempo mientras el aplicativo exista y posea actualizaciones, por lo que es relevante contar con un buen control de ellos en cuanto al alcance de cada uno, para lo cual se debe llevar un registro de cada caso en diferentes niveles:

- **A nivel de caso de prueba**

  Cada caso debe estar correctamente identificado, detallando, como mínimo, los siguientes campos:

  - **Producto**: Nombre del aplicativo para el cual fue desarrollado.
  - **Nombre del caso**: Etiqueta única del caso que debe ser la misma del caso manual de origen.
  - **Funcionalidad**: Acción que pretende certificar dentro del aplicativo.
  - **Subfuncionalidad** (cuando aplique): Submenú dentro de una funcionalidad.
  - **Estado**: Sus principales estados deben ser Automatizado (caso listo y funcional), En Construcción, En Mantención, Obsoleto.

  Si bien, cada vez que existe una liberación de un sistema que ha sufrido modificaciones se ejecutan los mismos casos regresivos, vale la pena saber cuáles son aquellos que impactan directamente en los cambios, con el objetivo de centrar la atención en ellos. Mediante esta clasificación, la identificación se torna una labor menos compleja, evitando la inspección de código y eliminando el retrabajo.

- **A nivel de ejecución**

  Cada vez que se ejecute un *set* de casos, se debe contar con un detalle histórico de la cantidad de casos fallados y pasados. Esto servirá para obtener métricas respecto de la calidad de la automatización y/o de la estabilidad del aplicativo, así como también de las causas más probables y constantes de fallas. Con esto, a medida que pasa el tiempo, se podrán crear ambientes estables y ejecuciones cada vez más confiables.

- **A nivel de *set* de pruebas**

  Los casos creados deben estar organizados en rutas dentro de un servidor o repositorio en donde todos los involucrados del proyecto puedan tener acceso. Estas rutas deben ser lo suficientemente descriptivas, con el fin de identificar claramente al proyecto al que pertenecen. En esta ruta deben estar los casos de prueba automáticos y los manuales.

Para todos los puntos mencionados existen herramientas que permiten la gestión y ejecución de casos de prueba. La decisión de utilizar una o no, queda a criterio de los involucrados, sin embargo, cuando se vaya a optar por una, se deben considerar algunos requisitos mínimos que permitirán llevar una buena administración de los *scripts*:

Tabla 8: Requisitos para evaluar una herramienta de gestión de casos de prueba
Fuente: Elaboración Propia

		Requisitos mínimos	Requisitos óptimos
Gestión	Plan de pruebas	✓	✓
	Caso de prueba	✓	✓
	Trazabilidad de casos	✗	✓
	Ejecución	✗	✓
Defectos	Reporte	✓	✓
	Historial de defectos	✓	✓
Reportes	Informe	✓	✓
	Dashboard	✗	✓
	Evidencia	✓	✓

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

El modelo propuesto es un modelo transversal, por lo que no requiere de un rubro empresarial específico para su aplicación. Bajo esta perspectiva y considerando la premisa del planteamiento de la solución, es que se realiza la validación en la empresa Finix Services, cuyo rubro principal es la prestación de servicios financieros.

Finix Services es una empresa nacional que comenzó prestando servicios en el año 2012, en ese entonces era operada por su dueño y un informático, su clientela era reducida, no superaba 4 clientes y operaba en una oficina pequeña ubicada en la comuna de Providencia. En el año 2013 la empresa logró aumentar su clientela de 4 a 8 clientes lo cual conllevó a un aumento del personal, principalmente contadores y a un cambio de oficina. Rápidamente para el año 2014 la empresa aumenta nuevamente su clientela lo que implicó un aumento de personal y un cambio de oficina a Av. Presidente Kennedy 9070 donde operaban 5 contadores, un informático y el dueño. Actualmente y dado a las circunstancias y necesidades del mercado, la empresa continua en crecimiento por lo que el 30 de octubre del 2017 fue necesario un nuevo cambio de oficina, esta vez con mayor espacio para el personal actual y espacio de sobra para satisfacer las proyecciones de crecimiento de un futuro cercano. La nueva oficina con un área de producción compuesta por 7 contadores y un ingeniero civil industrial, un área de desarrollo compuesta por 2 informáticos y un ingeniero civil industrial, una secretaria y el dueño quien tiene el puesto de gerente comercial.

Los principales servicios prestados por Finix Services son la valorización de fondos de inversión; realización de flujos de caja, reportes, pagos, balances de fondos de inversión; generación de estados de cuenta de aportantes y de estados financieros de fondos de inversión para empresas administradoras de fondos de inversión o AGF. Complementario a lo mencionado, Finix Services proporciona a sus clientes un portal de registro de operaciones, el cual se ajusta a las necesidades de los clientes por lo que se encuentra en un desarrollo constante. Este mismo portal es un gran canal de comunicación con los clientes, ya que es por este medio que los contadores suben gran parte de la información día a día, como también la AGF ingresa los nuevos aportantes de sus respectivos fondos. Un actor principal dentro de este portal es el usuario BackOffice, que es el encargado de dar soporte administrativo a los inversionistas. Dentro de sus servicios se encuentran:

1. **Valorización de fondos de inversión:** Este servicio es el principal de la empresa y consiste en calcular el valor cuota del fondo, considerando los movimientos que éste tuvo. Estos movimientos son inversiones según la naturaleza del fondo y lo que el reglamento interno de éste permita y gastos también ajustados a lo que el

reglamento establezca. Todos los movimientos se registran en el sistema de gestión de portafolio Geneva. Dentro de este mismo sistema se procede a generar una serie de reportes que se envían junto con la valorización como, por ejemplo, reporte de pérdidas y ganancias, activos y pasivos y balances.

2. **Generación de estados financieros**: El área de control normativo de la empresa realiza la confección de los estados financieros de los fondos. Para el caso de los fondos públicos, éstos son trimestrales, de carácter obligatorio y se deben publicar a la Comisión para el Mercado Financiero (CMF).

3. **Realización de reportes:** De manera complementaria a la valorización del fondo de inversión, para algunos fondos se desarrollan reportes de controles de límites, estos límites son definidos en el reglamento interno del fondo y principalmente tienen relación con la diversificación del riesgo y la liquidez del fondo.

4. **Circular NCG-364:** Circular de carácter obligatorio que deben entregar a la CMF todas las administradoras de fondos de inversión privado por cada fondo de manera trimestral. Es un archivo XML que se genera y contiene información del listado de partícipes del fondo, valores de activo y pasivos.

5. **Generación de estados de cuentas:** Según la periodicidad del fondo, se genera una cartola por cada inversionista o aportante con información de la inversión que posee, considerando el número de cuotas y la valorización calculada. Se genera y envía de manera masiva a cada aportante.

6. **Pagos:** Este servicio consiste en realizar algunos pagos como por ejemplo el pago de dividendos distribuidos por el fondo a cada aportante según su posición, pago de impuestos, pago de abogados y pago de auditorías. En algunos casos, cuando el fondo posee saldo disponible en su cuenta corriente y éste no será invertido en la naturaleza del fondo en el corto plazo, se procede a tomar inversión de fondo mutuo o depósito a plazo.

7. **Portal de aportantes:** Este servicio consiste en un portal web donde se encuentra la información de cada aportante, cabe destacar que el registro de aportante es obligatorio para los fondos de inversión y éste debe estar actualizado por la administradora del fondo respectivo. En este mismo portal se ingresa la valorización y el inversionista y BackOffice puede entrar a ver el estado actualizado de su inversión.

8. **Portal de Operaciones:** Este servicio es un portal web donde el usuario BackOffice puede entrar a registrar los movimientos de inversión que el fondo ha realizado, ya sea de renta variable o renta fija. El tipo de transacción que ingresa es compra, venta o abono. Con este movimiento ingresado se procede a realizar la gestión contable de la valorización. Cabe destacar que solo algunos clientes utilizan este portal de operaciones, el cual es un medio de comunicación. Los clientes que no utilizan este portal de operación envían la información vía correo electrónico en un formato determinado de Excel.

Los servicios descritos anteriormente se realizan en su totalidad a Administradoras de Fondos de Inversión y Administradoras General de Fondos. Actualmente la empresa presta servicios a 22 clientes que representan 121 fondos de inversión. Por motivos de confidencialidad no se entrega nombres de clientes.

4.1 CONTEXTO DE APLICACIÓN

La aplicación de este modelo se enfoca en la realización de reportes. El flujo comienza con la descarga del archivo Excel con los movimientos del cliente, se aplica cierto formateo a las columnas del documento para, finalmente, cargarlo al sistema que, en base a ciertos parámetros previamente ingresados, entrega el resumen con los datos requeridos.

Este aplicativo está constantemente sufriendo modificaciones de código con el objetivo de ajustarse a los requisitos de los clientes y entregar indicadores cada vez más precisos. Cuando hay una nueva liberación, la organización descarga una a una cierta cantidad de planillas de las páginas bancarias y las carga al aplicativo para verificar que los cambios realizados cumplen la solicitud inicial, sin embargo, el principal problema es que esta descarga de archivos es una tarea laboriosa no exenta del error humano que, en muchas ocasiones, demanda demasiado tiempo en comparación a la envergadura del requerimiento.

Teniendo esto, a priori se puede evidenciar que se está en presencia de un buen escenario a automatizar, sin embargo, es necesario someterlo a la aplicación del modelo para evitar la utilización de tiempo en generación de *scripts* que puede que, en un futuro, sean innecesarios.

Cabe señalar que, para efectos prácticos, la aplicación de este modelo es realizado a baja escala, sin embargo, la generalidad de sus definiciones permite aplicarlo a cualquier proyecto independiente de su envergadura.

4.2 IDENTIFICACIÓN

Se comienza con la identificación de los casos que se utilizan para realizar la certificación del aplicativo que, si bien no cumplen con los requisitos mínimos que se requieren para describir un correcto y claro caso de prueba, son suficientes para completar la tabla de ponderación. Los casos disponibles son sintetizados en Tabla 9.

Tabla 9: Casos de prueba
Fuente: Finix Services

ID	Caso	Descripción	Detalle
1	Cartola BCH	Descargar cartola de movimientos en Banco de Chile, cambiar formato y cargar al sistema	Dar clic en seleccionar filial y FONDO DE INVERSION PRIVADO TOESCA-I, luego cuentas corrientes, saldo y movimientos, seleccionar moneda y cuenta, si tiene más de una realizar n veces el proceso. Descargar en Excel. Luego modificar columna movimientos y cargar al sistema.
2	Cartola BICE	Descargar cartola de movimientos en banco BICE, cambiar formato y cargar al sistema	Prueba con los fondos que dicen Frontal trust y usa Cartola anterior. Luego modificar columna movimientos y cargar al sistema.
3	Cartola Santander	Descargar cartola de movimientos en banco Santander, cambiar formato y cargar al sistema	Probar fondo Volcomcapital Deuda Privada FIP y/o Voclom capital Deuda II Fondo de inversión, descargar Excel de la pestaña consulta de movimientos y la cartola provisoria. Luego modificar columna movimientos y cargar al sistema.
4	Reportes	Información del menú reportes	Ingresar al menú reportes y verificar información correcta
5	Cálculo NAV	Opciones cálculo NAV	Verificar todas las opciones, excepto opción grabar en opción de menú Cálculo NAV
6	Descargar y cargar	Descargar información	Descargar y cambiar formato descarga

Estos casos fueron la base para construir la Tabla de Ponderación que arroja el siguiente resultado:

Tabla 10: Tabla de Ponderación Aplicada
Fuente: Elaboración Propia

Índice	Parámetro	Evaluación
1	Criticidad	1
2	Portabilidad	0
3	Perfilamiento	1
4	Frecuencia	1
5	Complejidad	0

Con esta evaluación y la ponderación que corresponde a cada ítem, se tiene un resultado de 0.75, con lo que se concluye que éste es un escenario óptimo para automatizar, por lo que se procede a verificar cuáles son los casos que son susceptibles de automatización. Para esto se utiliza el Proceso de Identificación de Pruebas Automáticas, el cual entrega en su primera etapa el resultado que se muestra a continuación:

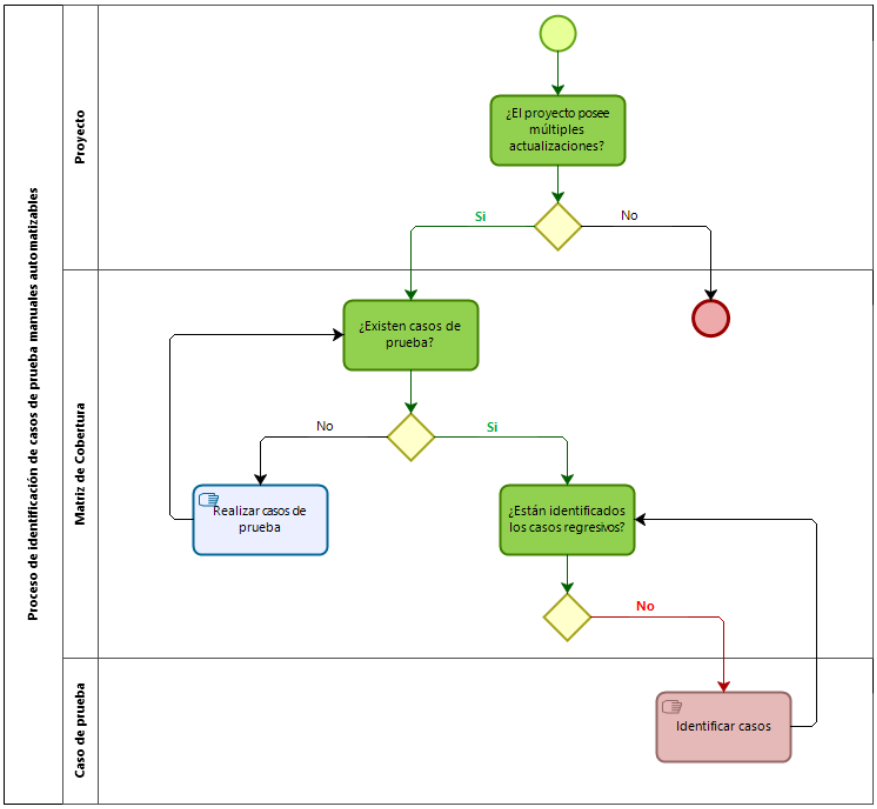


Figura 22: Proceso de Identificación de Pruebas Automáticas – Primera Etapa
Fuente: Elaboración Propia

La organización no tenía identificados cuáles de los casos son utilizados constantemente para verificar que los cambios realizados no afectan el normal funcionamiento del sistema, por lo que, los casos presentados anteriormente (Tabla 8), son sometidos a discusión entre el desarrollador y el encargado de las pruebas, obteniendo el siguiente filtro:

Tabla 11: Definición de casos regresivos
Fuente: Elaboración Propia

ID	Caso	Descripción	Detalle
1	Cartola BCH	Descargar cartola de movimientos en Banco de Chile, cambiar formato y cargar al sistema	Dar clic en seleccionar filiar y FONDO DE INVERSION PRIVADO TOESCA-I, luego cuentas corrientes, saldo y movimientos, seleccionar moneda y cuenta, si tiene más de una realizar n veces el proceso. Descargar en Excel. Luego modificar columna movimientos y cargar al sistema.
2	Cartola BICE	Descargar cartola de movimientos en banco BICE, cambiar formato y cargar al sistema	Prueba con los fondos que dicen Frontal trust y usa Cartola anterior. Luego modificar columna movimientos y cargar al sistema.
3	Cartola Santander	Descargar cartola de movimientos en banco Santander, cambiar formato y cargar al sistema	Probar fondo Volcomcapital Deuda Privada FIP y/o Volcom capital Deuda II Fondo de inversión, descargar Excel de la pestaña consulta de movimientos y la cartola provisoria. Luego modificar columna movimientos y cargar al sistema.

Los últimos 3 eran casos particulares que solo afectaban a las modificaciones de la última actualización, por lo que quedan fuera del conjunto de casos a automatizar, quedando solo los que son reciclados en cada liberación nueva. Posterior a esto, se continúa con el resto de las tareas del proceso.

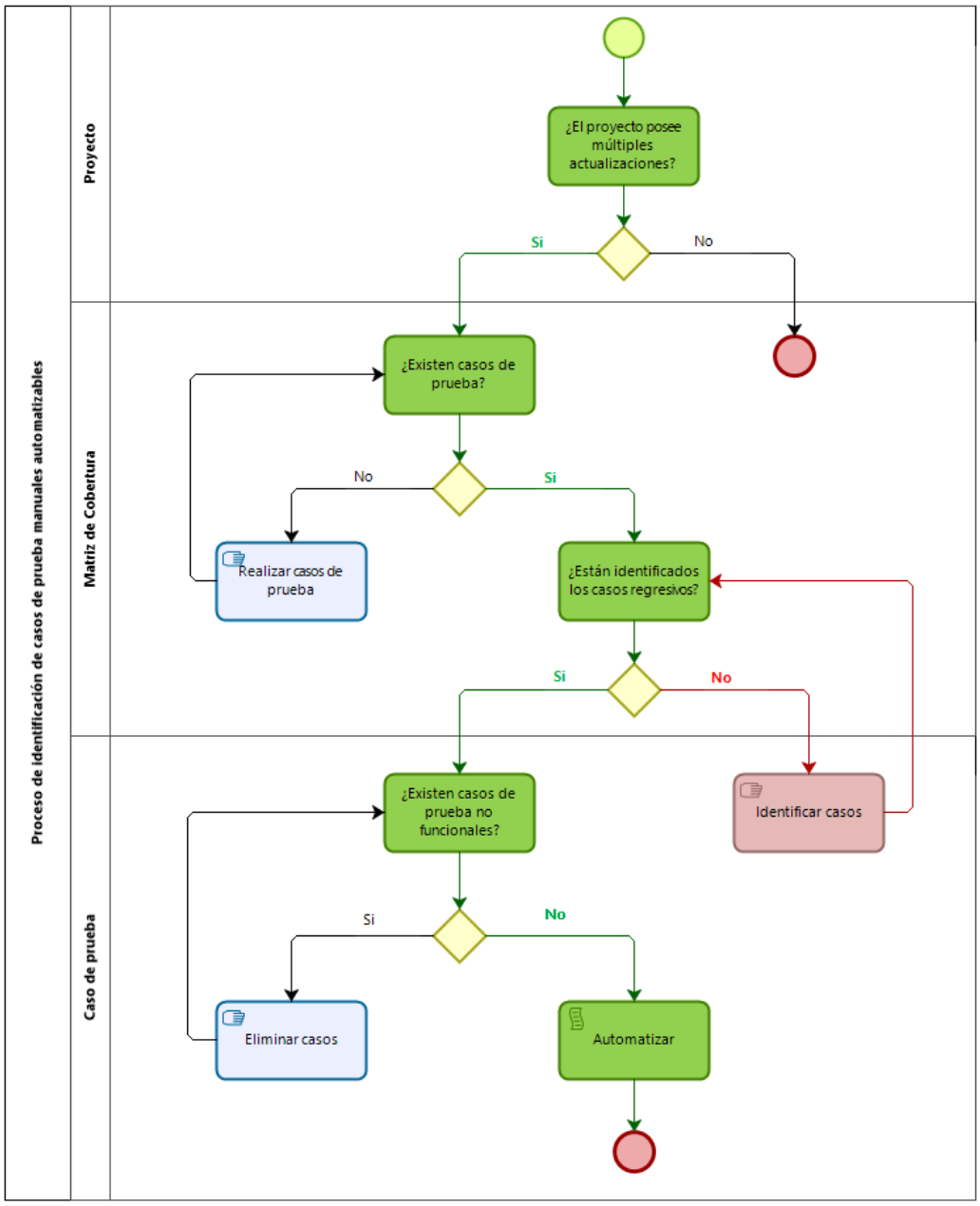


Figura 23: Proceso de Identificación de Pruebas Automáticas – Segunda Etapa
Fuente: Elaboración Propia

4.3 ANALISIS Y DISEÑO

En esta fase es el automatizador quien depura la salida de la etapa de Identificación sin intervención de los dueños del producto, existiendo 3 casos que se deben someter a evaluación para verificar si son sujetos de Modularización.

- **Caso 1: Cartola BCH**

Tabla 12: Predefinición de caso automático BCH
Fuente: Finix Services

ID	Caso	Descripción	Detalle
1	Cartola BCH	Descargar cartola de movimientos en Banco de Chile, cambiar formato y cargar al sistema	Dar clic en seleccionar filiar y FONDO DE INVERSION PRIVADO TOESCA-I, luego cuentas corrientes, saldo y movimientos, seleccionar moneda y cuenta, si tiene más de una realizar n veces el proceso. Descargar en Excel. Luego modificar columna movimientos y cargar al sistema.

Esta se realiza emulando de forma manual el comportamiento de lo que sería la automatización de la prueba en una estación de trabajo, concluyendo que éste es aplicable para modularización, pudiendo ser divido en 3 subcasos. El primero corresponde a la descarga de la cartola, para lo cual es necesario definir con claridad el paso a paso para facilitar la automatización:

1. Ingresar a la página del banco con credenciales empresa.
2. Seleccionar opción “Selección de Filiales” y luego fondo a descargar.
3. Seleccionar opción “Cuentas Corrientes” y posteriormente “Saldos y Movimientos”.
4. Seleccionar moneda y número de cuenta y presionar botón Consultar.
5. Presionar botón “Exportar Datos” y finalmente seleccionar la opción “Excel”.

El segundo corresponde al cambio de formato de las columnas para su posterior carga de acuerdo a los parámetros establecidos por la empresa, el cual no se detalla por políticas de privacidad de la organización. Finalmente, queda el caso de carga de documento, el

que consiste solo en ingresar al aplicativo y seleccionar la opción de carga de documento banco.

Estos últimos pueden ser clasificados como Genéricos, puesto que, como todas las cartolas que se descargan de la página del Banco de Chile poseen el mismo formato, cada vez que se descargue una cartola de un nuevo fondo, el cambio de formato y carga siempre será el mismo, por lo que solo se reutilizará el código para optimizar tiempo y asegurar un comportamiento uniforme en todas las acciones.

- **Caso 2: Cartola BICE**

Tabla 13: Predefinición de caso automático BICE
Fuente: Finix Services

ID	Caso	Descripción	Detalle
2	Cartola BICE	Descargar cartola de movimientos en banco BICE, cambiar formato y cargar al sistema	Prueba con los fondos que dicen Frontal trust y usa Cartola anterior. Luego modificar columna movimientos y cargar al sistema.

Este caso tiene el mismo análisis que el anterior, quedando subdivido en 3 subcasos: descarga, cambio formato y carga. El de carga se comporta de la misma manera como se mencionó en el caso 1, pero no así el cambio de formato, que tiene un trato particular para este banco. Para la descarga se define el siguiente paso a paso:

1. Ingresar a la página del banco con credenciales empresa.
2. Desplegar combobox de fondos y seleccionar fondo a descargar.
3. Presionar opción “Cuentas Pesos” y posteriormente “Cartola Provisoria”.
4. Seleccionar cuenta (cuando corresponda) y luego presionar botón Consultar.
5. Presionar botón “Exportar Excel”.
6. Presionar opción “Cuentas Pesos” y posteriormente “Cartola Anterior”.
7. Seleccionar cuenta (cuando corresponda) y luego presionar botón Consultar.
8. Presionar botón “Exportar Excel”.

- **Caso 3: Cartola Santander**

Tabla 14: Predefinición de caso automático Santander
Fuente: Finix Services

ID	Caso	Descripción	Detalle
3	Cartola Santander	Descargar cartola de movimientos en banco Santander, cambiar formato y cargar al sistema	Probar fondo Volcomcapital Deuda Privada FIP y/o Volcom capital Deuda II Fondo de inversion, descargar excel de la pestaña consulta de movimientos y la cartola provisoria. Luego modificar columna movimientos y cargar al sistema.

Similar a los anteriores con 3 subcasos, pero aplicados al banco Santander. Para la descarga se define el siguiente paso a paso:

1. Ingresar a la página del banco con credenciales empresa.
2. Seleccionar opción de fondo a descargar y presionar botón Ingresar.
3. Seleccionar opción “Cuentas Corrientes” y posteriormente “Cartola Provisoria”.
4. Presionar botón “Descargar” y seleccionar opción “Excel”.

Finalizada esta etapa, se evidencia el resultado en la siguiente tabla resumen:

Tabla 15: Definición de casos automáticos
Fuente: Elaboración propia

Caso	Nombre	Descripción
1	Descarga cartola Banco de Chile	Descargar cartola de movimientos en Banco de Chile
2	Cambiar formato Banco de Chile	Cambiar formato a columnas de movimientos de acuerdo a parámetros de empresa
3	Descarga cartola Banco BICE	Descargar cartola de movimientos en Banco BICE
4	Cambiar formato Banco BICE	Cambiar formato a columnas de movimientos de acuerdo a parámetros de empresa
5	Descarga cartola Banco Santander	Descargar cartola de movimientos en Banco Santander
6	Cambiar formato Banco Santander	Cambiar formato a columnas de movimientos de acuerdo a parámetros de empresa
7	Carga de archivo	Cargar archivo al aplicativo

La carga de archivo es una funcionalidad que tiene el mismo comportamiento independiente del banco con el que se esté trabajando, por lo que se define un solo caso que cubre esta acción.

Por decisión de los involucrados se decide realizar la automatización de las descargas de los 3 bancos como una primera etapa y dependiendo de sus resultados, se decidirá si se continúa con la automatización de los flujos restantes, por lo que se genera la matriz de cobertura para estos flujos de acuerdo a las definiciones de un caso de prueba manual:

Tabla 16: Detalle de casos automáticos
Fuente: Elaboración propia

ID	Nombre del caso	Descripción	Precondiciones	Datos de Prueba	Paso a paso	Resultado esperado
1	Cartola BCH	Descargar cartola de movimientos en Banco de Chile	Deben existir movimientos en cartola de cliente	*Se omiten por temas de confidencialidad*	1.- Ingresar a la página del banco con credenciales correctas 2.- Selecciona opción "Selección de Filiales" y luego fondo a descargar 3.- Seleccionar opción "Cuentas Corrientes" y posteriormente "Saldos y Movimientos" 4.- Seleccionar moneda y número de cuenta y presionar botón "Consultar" 5.- Presionar botón "Exportar Datos" y finalmentes seleccionar la opción "Excel"	Sistema debe descargar cartola del banco en formato excel en carpeta Descargas
2	Cartola BICE	Descargar cartola de movimientos en Banco BICE	Deben existir movimientos en cartola de cliente	*Se omiten por temas de confidencialidad*	1.- Ingresar a la página del banco con credenciales correctas 2.- Desplegar combobox de fondos y seleccionar fondo a descargar 3.- Presionar opción "Cuenta Pesos" y posteriormente "Cartola Provisoria" 4.- Seleccionar cuenta (cuando corresponda) y luego presionar botón "Consultar" 5.- Presionar botón "Exportar Excel" 6.- Presionar opción "Cuenta Pesos" y posteriormente "Cartola Anterior" 7.- Seleccionar cuenta (cuando corresponda) y luego presionar botón "Consultar" 8.- Presionar botón "Exportar Excel"	Sistema debe descargar cartola del banco en formato excel en carpeta Descargas
3	Cartola Santander	Descargar cartola de movimientos en Banco Santander	Deben existir movimientos en cartola de cliente	*Se omiten por temas de confidencialidad*	1.- Ingresar a la página del banco con credenciales correctas 2.- Selecciona opción de fondo a descargar y presionar botón "Ingresar" 3.- Seleccionar opción "Cuentas Corrientes" y posteriormente "Cartola Provisoria" 4.- Seleccionar opción "Descargar" y luego "Excel"	Sistema debe descargar cartola del banco en formato excel en carpeta Descargas

De acuerdo a los alcances y los recursos disponibles para esta actividad, es que la organización propone ciertos criterios para optar por la herramienta de automatización adecuada, en donde se establece que la herramienta:

1. Debe ser gratuita.
2. Debe soportar lenguaje Java y Python, que son los lenguajes utilizados por la organización.
3. Debe poseer documentación de fácil acceso.

Con base en estos requisitos, se decide utilizar Selenium que, aparte de cumplir con los dos primeros puntos, es un *framework* disponible desde el 2004, por lo que existe bastante información en la web donde se detalla cada una de las acciones que permite realizar, lo que facilitaría, en un futuro, continuar con las automatizaciones de los casos restantes si es que la organización decida retomar la actividad de forma independiente.

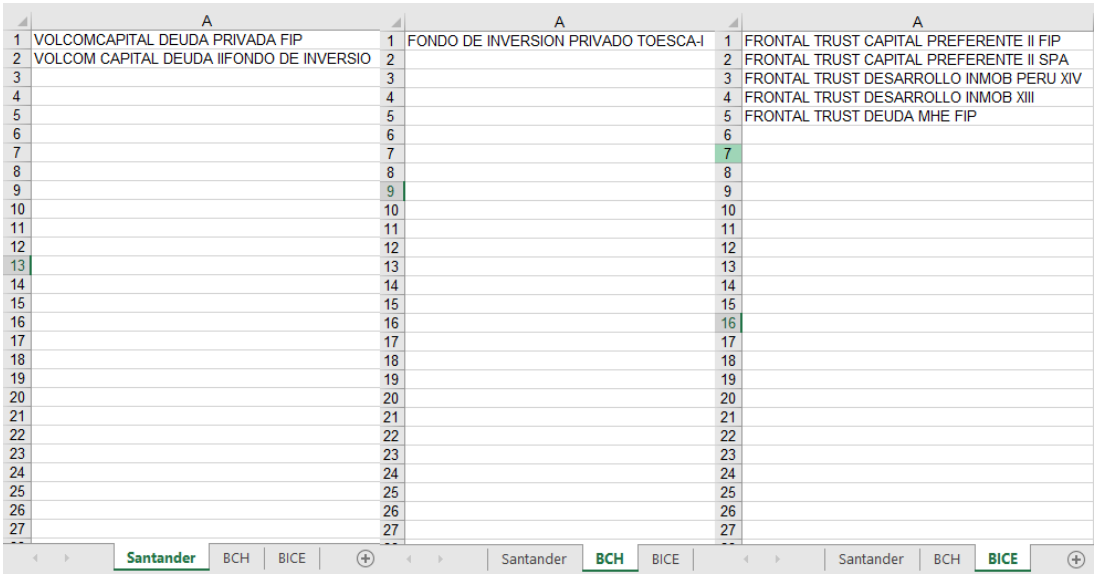
Para el proceso de desarrollo se consideran los 5 principios de desarrollo de casos automáticos:

1. **Comentarios**: Se crean comentarios aclarativos en algunas sentencias con el fin de facilitar el entendimiento el código.

```
//Recorrer el archivo de fondos
int filas = sheet.getLastRowNum();
for (int i = 0; i <= filas; ++i) {

    HSSFRow row = sheet.getRow(i);
    HSSFCell celdaFondo = row.getCell(0);
    String fondo = celdaFondo.getStringCellValue();

    //Buscar si el fondo es el primero en la lista del banco
    if(!(Selenium.EncontrarElemento("//b[contains(text(),'"+fondo+"')]"))) {

        cantidad_elementos = Selenium.ContarElementos("(//input[@type='radio'])");

        //Buscar la posición donde se encuentra el fondo buscado
        while( j <= cantidad_elementos ){

            fondo_encontrado = Selenium.BuscarTexto("((//form//table)[4]//a)["+j+"]");
```

Figura 24: Comentarios en código
Fuente: Elaboración Propia

2. **Parametrización:** Se crea archivo Excel llamado “DetalleFondos.xls”, en donde cada banco posee una hoja distinta con los nombres de los fondos que se deben descargar por cada uno. Con esto, cada vez que se deseen agregar o eliminar fondos, solo debe modificar el archivo sin intervenir el código fuente.



	A
1	VOLCOMCAPITAL DEUDA PRIVADA FIP
2	VOLCOM CAPITAL DEUDA IIFONDO DE INVERSIO

Santander | BCH | BICE

	A
1	FONDO DE INVERSION PRIVADO TOESCA-I

Santander | BCH | BICE

	A
1	FRONTAL TRUST CAPITAL PREFERENTE II FIP
2	FRONTAL TRUST CAPITAL PREFERENTE II SPA
3	FRONTAL TRUST DESARROLLO INMOB PERU XIV
4	FRONTAL TRUST DESARROLLO INMOB XIII
5	FRONTAL TRUST DEUDA MHE FIP

Santander | BCH | BICE

Figura 25: Contenido archivo DetalleFondos.xls
Fuente: Elaboración Propia

3. **Control de errores:** Por cada función se crea el control *try catch*, en donde cada error es controlado con un mensaje claro y entendible por el ejecutor del caso.

```
public String BuscarTexto(String RutaXpath){
    // Iniciamos la variable para almacenar el texto leido.
    String Lectura = null;
    Object logger;
    try {
        // Hacemos una breve pausa antes de realizar alguna accion.
        Thread.sleep(1000);

        // Buscamos el elemento por el xpath indicado.
        Elemento = Driver.findElement(By.xpath(RutaXpath));

        // Ingresamos el texto indicado en el elemento.
        Lectura = Elemento.getText().trim();
    }

    catch (Exception Ex) {
        // En caso de error arrojamos un mensaje.
        ErrorJava("Hubo un error al leer el texto en el elemento con la ruta: " + RutaXpath, Ex);
        String screenshotPath = GetScreenshot.capture(Driver,"Error al buscar texto");
        logger.log(LogStatus.FAIL, logger.addScreenCapture(screenshotPath));
    }

    // Devolvemos el texto encontrado.
    return Lectura;
}
```

Figura 26: Control de errores
Fuente: Elaboración Propia

4. **Independencia:** El flujo de descarga de cada banco se crea dentro un test distinto, lo que se observa en la figura 27 con la etiqueta @Test. Esto permite saber, en caso de falla, cuál es el flujo que presenta problemas u omitir alguno de ellos cuando no se deseen ejecutar todos. Esto último se realiza comentando el test a excluir.

   Cada caso inicia el *driver* que permite el manejo del navegador y lo finaliza cuando termina su ejecución.

```
@Test // Descarga de informes Santander
public void DescargaSantander() throws InterruptedException, FileNotFoundException, IOException {

    Driver = Selenium.IniciarDriver("https://www.santander.cl/empresas/index.asp");
    Santander FlujoSantander = new Santander();
    FlujoSantander.Descarga();
    Selenium.FinalizarDriver(Driver);
}

@Test // Descarga de informes BICE
public void DescargaBICE() throws InterruptedException, FileNotFoundException, IOException {

    Driver = Selenium.IniciarDriver("https://bice.cl/");
    BICE FlujoBice = new BICE();
    FlujoBice.Descarga();
    Selenium.FinalizarDriver(Driver);
}

@Test // Descarga de informes BCH
public void DescargaBCH() throws InterruptedException, FileNotFoundException, IOException {

    Driver = Selenium.IniciarDriver("https://ww3.bancochile.cl/wps/wcm/connect/bch-empresas/bancodechile/empresas");
    BancoChile FlujoBCH = new BancoChile();
    FlujoBCH.Descarga();
    Selenium.FinalizarDriver(Driver);
}
```

Figura 27: Independencia de casos
Fuente: Elaboración Propia

5. **Evidencia:** Se crea una función que captura *screenshots* cada vez que ésta es ejecutada, la cual es llamada dentro del control de errores. Con esto se permite evidenciar, de manera gráfica, el motivo de la falla.

```
public class GetScreenshot {

    public static String capture(WebDriver driver, String screenshotName) throws IOException {
        TakesScreenshot ts = (TakesScreenshot)driver;
        File source =ts.getScreenshotAs(OutputType.FILE);
        String dest = "C:\\Reportes\\Evidencia\\"+screenshotName+".png";
        File destination = new File(dest);
        FileUtils.copyFile(source, destination);

        return dest;
    }
}
```

Figura 28: Función de *screenshots*
Fuente: Elaboración Propia

4.4 EJECUCIÓN

La ejecución se realiza dentro de una máquina virtual habilitada para este fin, la cual considera los siguientes requisitos para su creación:

- RAM: 4GB
- HDD: 10GB
- Sistema Operativo: Windows 10 64bits

Las pruebas se ejecutan sin problemas, sin intervención humana y mostrando un resultado exitoso en el 100% de las ejecuciones. Es importante destacar que, independiente de este resultado, no se puede asegurar que en cada ejecución se obtendrá el mismo porcentaje de éxito, debido a que las páginas del banco están sujetas a cambios que es imposible controlar si no se tiene el detalle de ese cambio: el banco puede agregar ventanas emergentes promocionales a su página de inicio, por ejemplo, lo que haría que el *login* falle, por lo que es importante estar pendiente de estos cambios para tomar acciones proactivas en el momento oportuno y minimizar el impacto.

El tiempo que demora la ejecución de los 3 flujos de descarga de documentos es de 7 minutos en promedio, lo cual supera la prueba manual en alrededor de 2 minutos, sin embargo, esto se explica por la parametrización de fondos: el *script* toma el primer fondo de la lista de “DetalleFondos.xls” y compara uno a uno en la lista de fondos disponibles en la página del banco, lo que toma un tiempo mayor a que si lo realizara un *tester* de manera manual. Luego continúa y realiza lo mismo con los “n” fondos que se listen en el documento Excel.

4.5 CONTROL

Siempre es recomendable utilizar una herramienta especializada que permita realizar un buen *tracking* de la ejecución de los casos, con esto se mantiene el orden y permite una buena gestión del conocimiento ante eventuales rotaciones de personal o modificaciones/mantenciones posteriores. Para este fin y a modo exploratorio, se decide usar TestLink, un aplicativo gratuito que permite hacer la gestión básica de las automatizaciones y proporciona una escalabilidad de cantidad de casos y ejecuciones sin límites.

TestLink, al ser instalado, genera un usuario administrador con el que permite realizar toda la configuración del aplicativo para ser utilizado por los involucrados en el proyecto. Con éste se crea, en primera instancia, el nombre del proyecto: "Descarga Cartolas Project". Posteriormente se crean los dos usuarios que serán los encargados de gestionar el proyecto y los casos de prueba: "leader" y "tester".

- **Leader:** Este usuario es el encargado de crear los casos de pruebas que serán ejecutados en el proyecto, los cuales siguieron el mismo detalle presentado en la Tabla 16. En este caso en particular, cada caso de prueba queda asociado a un *Test Suite*[3] distinto por el alcance que tiene la automatización, sin embargo, el objetivo es que se vayan agregando casos a medida que el proyecto crezca en magnitud. Los 3 *Test Suite* distintos son: "Descargas de cartola BCH", "Descargas de cartola BICE" y "Descargas de cartola Santander". Este rol también tiene como tarea crear el *Test Plan*[4] de la certificación, el cual posee el nombre de "Descarga Cartolas TP" y contiene los 3 casos de la Tabla 16.

- **Tester:** Este usuario es el encargado de ejecutar los casos de prueba creados en el *Test Plan*, debiendo asignar el resultado de cada ejecución realizada.

Finalizada la configuración, el proyecto posee la siguiente estructura:

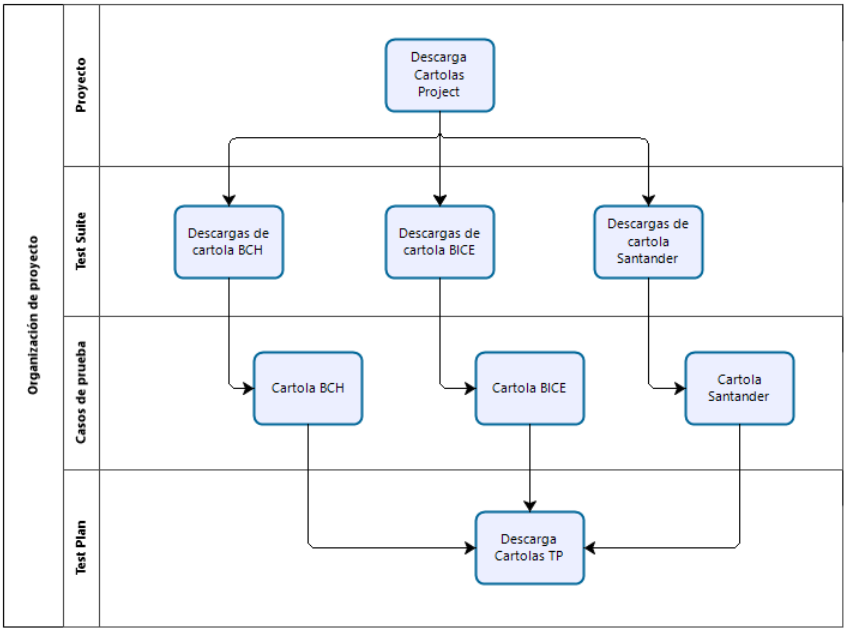


Figura 29: Diagrama de estructura del proyecto
Fuente: Elaboración Propia

[3] Conjunto de casos de prueba destinados a validar y verificar un comportamiento en común
[4] Conjunto de casos de prueba agrupados para realizar una certificación. Puede contener casos asociados a distintos *Test Suite.*

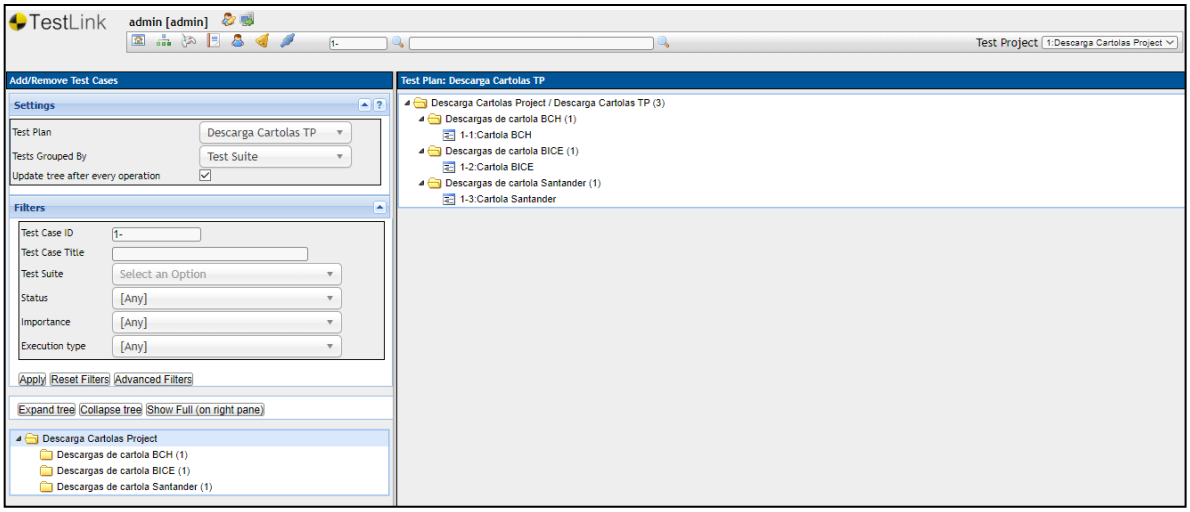


Figura 30: Estructura del proyecto en TestLink
Fuente: Elaboración Propia

CAPÍTULO 5: CONCLUSIONES

5.1 CONCLUSIONES GENERALES

Es bien sabido que la tecnología avanza a pasos agigantados, en muchas ocasiones a una velocidad que obliga a la sociedad a una rápida adaptación que no siempre es fácil de lograr, pero necesaria, debido a que actualmente la vemos aplicada en todo ámbito dentro de una comunidad, desde una actividad cotidiana como programar una alarma hasta acciones más complejas de procesamiento de información o control de enormes maquinarias a distancia. Sin embargo, para que esta tecnología cumpla su objetivo es importante asegurar que hará lo que promete hacer de la forma que se ofrece y he aquí donde radica la importancia de las pruebas de aseguramiento de calidad de software; su correcta aplicación proporciona un mayor grado de seguridad para su uso tanto para el proveedor como para el usuario.

Para todo desarrollo de productos, ya sea tangibles o intangibles, destinados a satisfacer las necesidades humanas se debe considerar la premisa que siempre estarán propensos a fallas que, en el peor de los casos, pueden dañar la integridad del usuario que los utiliza, más aún si están directamente relacionados con la salud de las personas. Es por esto que se debe poner especial énfasis en los sistemas de *software*, puesto que su cualidad de abstracto e intangible dificulta identificar todos los posibles escenarios que pueden hacer que éstos presenten algún comportamiento inesperado y cuantificar las eventuales consecuencias que pudiera ocasionar y, a diferencia de un producto tangible que se produce en serie, cuyos desperfectos descubiertos en alguno de ellos o en el proceso productivo, implican solo hacer un cambio en el evento identificado, el software es único y no puede ser desechado, por lo que debe mejorarse continuamente y, de cierta forma, no acepta tolerancia a fallas una vez liberado para su uso.

La calidad es un concepto muy transversal que aplica a cualquier producto que esté destinado a satisfacer una necesidad, característica que provoca que su interpretación y ámbito de aplicación sea muchas veces ambigua, poco clara o incluso desconocida y omitida, por lo que basarse en algún modelo es de gran ayuda al momento de querer implementar la calidad en el *software*, puesto que si no se realiza de forma ordenada y sistemática, es muy posible que la desorganización haga fracasar su implementación, incurriendo en pérdidas de recursos humanos y económicos. La causa principal de este desorden ocurre porque las tareas que deben ejecutar los involucrados están tan relacionadas e interconectadas, que muchas veces se confunde cuáles son las labores

específicas que debe ejecutar cada integrante del equipo, generando conflictos laborales, atrasos en entregas, descoordinación en las planificaciones, etc.

El hecho de que en los modelos se defina qué tareas realizar, la forma correcta de ejecutarlas y el orden de ellas, aumentan las probabilidades de terminar un proyecto de manera exitosa, siempre y cuando exista una clara organización y coordinación de cada uno de los integrantes del equipo.

Existen muchos modelos de calidad, tal como se describió en el capítulo 2 de este documento, por lo que cada compañía debe decidir el qué se debe utilizar dependiendo de cuál se ajuste mejor a su cultura organizacional, puesto que, si bien todos tienen un objetivo en común, no todos tienen el mismo ámbito de aplicación. Al momento de optar por alguno de ellos, se debe tener claro que muchas veces la teoría dista de la realidad, lo que repercute en que no pueda ser adaptado en un 100%, por lo que es común que se realice un proceso de ajuste de algunas de las propuestas que éste plantea en cuanto a la forma, pero cuidando de mantener el fondo, de tal manera de no alterar el sentido de la propuesta del modelo.

En particular, con el modelo de automatización de pruebas se vio reflejado que, al momento de presentarlo en la organización, existían muchas dudas sobre las tareas a realizar, las herramientas a utilizar, por dónde comenzar, qué abarcar, etc., por lo que el juicio experto se hizo relevante a la hora de implementar el modelo, guiar la ejecución de las tareas, reunir y discriminar la información, compendiar los datos, etc. Si bien, los modelos buscan organizar la forma de realizar las cosas, siendo de gran utilidad cuando se tiene poco conocimiento de cómo realizar un proceso en específico, es importante que exista un experto que guie y se dedique a su implementación; esto ayudará a minimizar los tiempos y los costos asociados.

Cabe señalar que cualquier modelo nuevo está sujeto a un proceso de maduración en base a la experiencia. Las pruebas automatizadas, basado en la realidad nacional, son relativamente nuevas en el mundo del SQA, por lo que cualquier modelo que se proponga en esta vía, estará sujeto a modificaciones u optimizaciones futuras a medida que vayan surgiendo nuevas herramientas y formas de trabajo, sin embargo, es bueno comenzar con una base, porque aporta la experiencia necesaria para llegar a la madurez requerida, aparte de entregar indicadores que sirven para implementar las mejoras y derribar mitos que surgen del desconocimiento y de la poca información que se tiene al respecto.

Uno de los principales mitos del que se habla cuando se toca el tema de las pruebas automáticas, es que el objetivo de ellas es que se ejecuten más rápido que las pruebas manuales. Esto no es del todo cierto, puesto que depende del contexto bajo el que se esté evaluando esta premisa y del aplicativo sobre el cual se vayan a correr las pruebas: puede cumplirse si se considera un conjunto de pruebas destinadas a una certificación que deben ser ejecutadas de manera secuencial, puesto que, si éstas se realizaran de forma manual, al tiempo que le tomaría a un *tester* ejecutarlas se le debe agregar un delta de tiempo asociado a las distracciones y agotamiento que provoca la realización de pruebas repetitivas, lo cual es directamente proporcional a la cantidad de casos a ejecutar. Ahora bien, si esto lo llevamos a pruebas automáticas, este delta de tiempo se elimina, lo que, en su totalidad, puede generar una disminución del tiempo total de ejecución. Sin embargo, esto no se puede asegurar con certeza, debido a que, tal como se describió en el capítulo 3, las buenas prácticas y recomendaciones al momento de generar un *script* como el control de errores, la independencia de los casos o la parametrización, pueden ocasionar que la prueba tome un tiempo mayor que el que le tomaría a un *tester* manual ejecutar, por lo que se debe ser muy cuidadoso a la hora de crearse las expectativas respecto a estas pruebas.

La implementación de la solución se realizó a menor escala, sin embargo, fue suficiente para evidenciar que no todos los casos son factibles de automatizar, puesto que de inmediato se descartaron aquellos relacionados con modificaciones particulares. Si bien se podrían haber realizado, no valía la pena invertir tiempo y dinero en cambios puntuales que, muy probablemente, no serían intervenidos dentro del corto plazo, siendo relegadas a pruebas de forma manual. Esto deja en evidencia que las pruebas automáticas no reemplazan a las pruebas manuales, sino que apuntan a ser un complemento que permite validar y verificar que las funcionalidades de un aplicativo que ha sido modificado sigan manteniendo estabilidad en cuanto al flujo feliz. Esto posee un gran impacto a nivel empresarial:

- Al ejecutar pruebas desatendidas, permite liberar recursos humanos pudiendo realizar otras labores, lo que aumenta la productividad de sus procesos.

- Permite optimizar los tiempos de pruebas, al poder ejecutar pruebas de manera desatendida y en ocasiones, fuera del horario laboral, programándose para ejecuciones nocturnas y evaluando resultados al día siguiente.

- Mejora la imagen de la organización frente a los clientes, entregando resultados en menor tiempo y más consistentes.

Considerando que las pruebas manuales son la base de las pruebas automáticas, se torna fundamental una correcta definición de éstas, tal como se mencionó en el capítulo 2 y plasmarlo en una matriz de cobertura (o matriz de casos de pruebas) como se realizó al momento de la validación. Cabe señalar que el creador esta matriz no necesariamente es el que debe automatizarlas, debido a que, a pesar de estar muy relacionadas entre sí, son tareas independientes, por lo que el automatizador no está obligado a tener un conocimiento acabado de las reglas del negocio, debiendo, cada prueba, quedar para cualquier persona que la lea. Esto hace que la descripción de cada caso deba ser lo más detallada y descriptiva posible para evitar retrabajo al momento de automatizar o incluso al realizar una certificación manual del mismo aplicativo.

La herramienta que se desee utilizar está supeditada a los recursos disponibles por la organización para estas tareas, pero es importante considerar que las pruebas aisladas, si bien generan resultados positivos para una organización, es de suma relevancia mantener un correcto control de estos casos, debiendo utilizar alguna herramienta que permita su gestión pensando que serán siempre útiles mientras el aplicativo existe y preste utilidad a la empresa. Su correcta administración permitirá tener mantenciones controladas, identificaciones más precisas y en general, un control de conocimiento que servirá para tomar decisiones basadas en datos empíricos apuntando a mejorar la calidad de los sistemas.

Como se pudo observar, el modelo propuesto solo está orientado a la transición de pruebas manuales a automáticas, pero desde ese punto hacia atrás hay muchas tareas que es importante que la organización considere si quiere entregar un producto de buena calidad, desde que nace la idea del dueño del producto hasta que finalmente es liberado, pasando por el levantamiento de requerimientos, análisis de documentación, generación de casos de pruebas, estrategias de pruebas y pruebas automatizadas. Para todos estos puntos existen sugerencias en la literatura que apoyan que estos procesos sean generados de una manera óptima, por lo que no es necesario reinventar la rueda y es responsabilidad de la organización interiorizarse en estos temas si se quieren obtener buenos resultados.

En síntesis, la implementación de un modelo como proyecto de mejora continua requiere de un proceso de adaptación que va directamente relacionado con un cambio cultural en la forma de realizar las tareas diarias. Siempre va a existir una resistencia al cambio por algunos miembros del equipo, puesto que el temor a lo desconocido es algo inherente al ser humano, pero es un tema que se debe conversar y trabajar, sobre todo cuando la orientación es la optimización de recursos, sin embargo, si bien se sabe que toma tiempo, es algo que se va aprendiendo con la práctica. Es por lo que, a pesar de que este modelo

tuvo resultados satisfactorios para Finix Services, necesita pasar por un periodo de prueba que puede tomar meses, en donde surgirán problemáticas que impulsarán cambios en las propuestas planteadas y posiblemente, crearán nuevos modelos que mejorarán lo que en este documento se ha planteado.

5.2 TRABAJO FUTURO

Mediante este trabajo se pudo verificar que el modelo propuesto es una buena herramienta de apoyo para una organización que desea realizar una transición ordenada desde pruebas manuales a pruebas automatizadas. Sin embargo, a pesar de haber sido aplicado a un proyecto con un alcance bien acotado, se pudo notar que existía falta de conocimiento en precondiciones que, para efecto de la creación del modelo, se asumieron inexistentes, más específicamente, se habla del tratamiento y manejo que se tenía de los casos de prueba manuales: no poseían un orden y estructura requeridas para una correcta gestión. Si bien esto se pudo solucionar con el conocimiento que se poseía en base a la experiencia, no todas las empresas tendrán la oportunidad de mitigar este riesgo.

Con base en lo expuesto anteriormente y como trabajo futuro, se propone crear un modelo más amplio que permita mitigar los cursos alternativos que no estén cubiertos con los conocimientos existentes en el contexto de aplicación. A priori, se mencionan los siguientes puntos:

- Estructura necesaria de las pruebas destinadas a la certificación para ser abordadas como pruebas manuales y cómo incluirla.
- Mejor estrategia de pruebas en base a la metodología utilizada para las certificaciones.
- Manejo de defectos encontrados en las validaciones y verificaciones.

Esta lista es un instrumento vivo que puede ir creciendo de acuerdo con el historial que se vaya generando en las múltiples implementaciones que surjan para distintas empresas, culturas, formas de trabajo y alcances de los proyectos, que tendrá como objetivo la creación de un modelo que parta desde que nace el proyecto hasta que es liberado a producción.

REFERENCIAS BIBLIOGRÁFICAS

[1] Alaqail, H. y Shakeel. A. (2018). *Overview of Software Testing Standard ISO/IEC/IEEE 29119*. IJCSNS International Journal of Computer Science and Network Security, vol. 18 (nro. 2)

[2] Callejas, M., Alarcón, A. y Álvarez, A. (2017). *Modelos de Calidad del Software, un Estado del Arte.* Revista Entramado, vol. 13 (nro. 1)

[3] Constanzo, M. A. (2014). *Comparación de Modelos de Calidad, Factores y Métricas en el Ámbito de la Ingeniería de Software*. Universidad Nacional de la Patagonia Austral. Río Gallegos. Argentina.

[4] Crisspin, L. y Gregory, J. (2009). *Agile Testing. A Practical Guide for Testers and Agile Teams.* Estados Unidos: Addison-Wesley.

[5] Cristiá, M. (2015). *Introducción al Testing Tradicional.* Universidad Nacional de Rosario. Rosario. Argentina.

[6] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Granada. *Accidentes Therac-25* [en línea]. <https://lsi.ugr.es/~mvega/docis/aluwork/roddesastres/therac.htm>. [Consulta: 18 de julio de 2019]

[7] Diacos Asesores (2017). *Testing Ágil. Primera Edición.* Santiago. Chile.

[8] Douglas, N. A. (1998). *The Patriot Missile Failure* [en línea] <http://www-users.math.umn.edu/~arnold//disasters/patriot.html>. [Consulta: 18 de julio de 2019]

[9] García, A. (2006). *Estudio Comparativo de los Modelos y Estándares de Calidad del Software.* Universidad Tecnológica Nacional, Facultad Regional Buenos Aires. Buenos Aires.

[10] González, A., Ampuero, M. y González, A. (2015). *Análisis Comparativo de Modelos y Estándares para Evaluar la Calidad del Producto de Software.* Revista Cubana de Ingeniería, vol. 6 (nro. 3)

[11] Gonzales, E. (1996). *El Ariane 5 explotó por un fallo de 'software' afirma el informe oficial.* El País [en línea] <https://elpais.com/diario/1996/07/24/sociedad/838159214_850215.html>. [Consulta: 18 de julio de 2019]

[12] González, L. (2009). *Método para Generar Casos de Prueba Funcional en el Desarrollo de Software.* Revista Ingeniería Universidad de Medellín, vol. 8 (nro. 15).

[13] IEEE Computer Society (2014). *IEEE Standard for Software Quality Assurance Processes*. New York, USA.

[14] IEEE Standards Association. *IEEE 1012-2004 – IEEE Standard for Software Verification and Validation* [en línea] <https://standards.ieee.org/standard/1012-2004.html>. [Consulta: 22 de abril de 2019]

[15] IEEE Standards Association. *IEEE 1061-1998 – IEEE Standard for a Software Quality Metrics Methodology* [en línea] <https://standards.ieee.org/standard/1061-1998.html>. [Consulta: 22 de abril de 2019]

[16] Institut Puig Castellar. *Prueba de aplicaciones web y documentación* [en línea] https://elpuig.xeill.net/Members/vcarceler/asix-m09/uf1/nf1/a5. [Consulta 07 de julio de 2019]

[17] International Software Testing Qualifications Board. *About ISTQB* [en línea] <https://www.istqb.org/about-as.html>. [Consulta: 25 de abril de 2019]

[18] ISO 25000. *La familia de normas ISO/IEC 25000* [en línea] <https://iso25000.com/index.php/normas-iso-25000>-. [Consulta: 22 de abril de 2019]

[19] ISTQB (2010). *Fundation Level Syllabus*. Bélgica.

[20] Joskowicz, J. (2008). *Reglas y Prácticas en eXtreme Programming*. Vigo, España.

[21] Mascheroni, M. A., Cogliolo, M. K. y Irarrazabal, E. A. (2016). *Automatización de Pruebas de Compatibilidad Web en un Entorno de Desarrollo Continuo de Software.* Universidad Nacional del Nordeste. Facultad de Ciencias Exactas y Naturales y Agrimensura. Argentina.

[22] Mera-Paz, J. A. (2016). *Análisis del Proceso de Pruebas de Calidad de Software.* Ingeniería Solidaria, vol. 12 (nro. 20).

[23] Moreno, J. J., Bolaños, L. P. y Navia, M. A. (2010). *Exploración de Modelos y Estándares de Calidad para el Producto de Software.* Universidad del Cauca. Colombia.

[24] Myers, G. J. (2004). *The art of Software Testing. Second Edition.* New Jersey, Estados Unidos: John Wiley & Sons, Inc.

[25] Pressman, R. S. (2010). *Ingeniería del Software – Un enfoque práctico.* Nueva York. Estados Unidos: McGRAW-HILL.

[26] Real Academia Española. *Diccionario de la lengua española – Edición del Tricentenario – Actualización 2018* [en línea] <https://dle.rae.es/?id=YErlG2H>. [Consulta: 05 de mayo de 2019]

[27] Real Academia Española. *Diccionario de la lengua española – Edición del Tricentenario – Actualización 2018* [en línea] <https://dle.rae.es/?w=calidad>. [Consulta: 05 de mayo de 2019]

[28] Sánchez, J. M. (2015). *Pruebas de Software. Fundamentos y Técnicas.* Universidad Politécnica de Madrid. Madrid.

[29] Standards Coordinating Comitee of the Computer Society of the IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology.* Nueva York. Estados Unidos.

ANEXOS

finixservices

Informe de verificación

ASUNTO:

Validación de memoria "Modelo de Transición de Desarrollo y Ejecución de Pruebas Manuales a Pruebas Automatizadas como Estrategia en el aseguramiento de Calidad del Software"

Los flujos automatizados funcionaron de forma correcta según lo esperado y se verifica que el alcance del requerimiento establecido al inicio del desarrollo ha cumplido en un 100% sobre el entorno de desarrollo (IDE).

Debido a las restricciones del IDE utilizado el robot solo se puede ejecutar en una sesión abierta y activa en modo gráfico, por lo que se propone como mejora el poder ejecutarlo como un servicio sin necesidad de tener una sesión abierta.

Por nuestra parte, se quiso implementar la creación de un ejecutable para independizar el desarrollo del IDE, que fue lo más complejo de realizar, por lo que también se propone aumentar los alcances hasta la conversión de las automatizaciones a un ejecutable.

Raimundo Ducci
Gerente
Finix Services

finixservices

Informe de verificación

ASUNTO:

Validación de memoria "Modelo de Transición de Desarrollo y Ejecución de Pruebas Manuales a Pruebas Automatizadas como Estrategia en el Aseguramiento de Calidad de Software"

Durante la implementación de la automatización de descarga de cartolas no se presentaron grandes inconvenientes en base al alcance definido en un inicio, la descarga la realizaba según lo planificado y acorde a lo indicado en la parametrización. Uno de los puntos a considerar y que podría generar inconvenientes en un futuro es la publicidad que algunas veces muestran los bancos dentro de la pagina web. En este caso se tendría que modificar la automatización lo cual probablemente significaría que se tendría que realizar el proceso manual por un periodo de tiempo hasta que se modifique y apruebe la nueva automatización. Si bien la automatización realiza la descarga de n cartolas para 3 bancos, estos solo guardan el archivo en la carpeta de descarga del navegador o en una carpeta específica, esto implica que si bien ya no será necesario descargar la cartola de manera manual si se tendrá que ir a buscarla a una carpeta especifica donde se tendrá las cartolas de todos los clientes de la empresa lo cual genera un riesgo de cruzar información entre clientes. Para eliminar este riesgo es que se propone para un futuro generar un proceso que lea todas las cartolas y las ingrese en la base de datos de la empresa para mostrarlas directamente en el sistema propio de conciliación de Finix, este último es un proceso manual que se realiza luego de descargar las cartolas.

Hoy en día la empresa presta alrededor de 16 servicios diarios, esto significa descargar alrededor de 50 cartolas diarias. En cuanto a tiempo gastado por parte del contador que realiza el servicio en total serían más de 1 hora diaria solo descargando cartolas, con la implementación total de esta automatización de descarga de cartolas se ahorraría todo este tiempo y permitiría al contador enfocarse netamente en el core del negocio eliminado estas tareas rutinarias.

Como punto a mejorar, se proponer poder seleccionar la carpeta de descarga según la cuenta, traspasar el ejecutable a una versión online con mantenedor para seleccionar las cartolas a descargar. Para las cartolas provisorias indicar si existe movimientos nuevos versus lo descargado anteriormente.

A modo de proyección en el tiempo, en el mediano plazo esta automatización la necesitamos poder implementar en su totalidad, es decir para el 100% de los servicios que prestamos se descargue la cartola del o los bancos de manera automática esto implica generar la automatización para todos los bancos, es un proceso que se aplicaría de manera nocturna para los servicios que poseen desface y de manera directa o unitaria para los servicios que se realizan con la información del mismo día. En un futuro también sería aplicable el mismo proceso a una descarga de cartolas de custodia de instrumentos financieros de la cual también se saca información para la conciliación y/o se procese a realizar cuadraturas versus lo que se mantiene en nuestros registros.

Antonio Vega
Analista de Operaciones
Finix Services