UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA DE SOFTWARE

Técnicas de pruebas de software más utilizadas por un Tester egresado de la UACM

MEMORIA DE EXPERIENCIA PROFESIONAL

PARA OBTENER EL TÍTULO DE

LICENCIADO EN INGENIERÍA DE SOFTWARE

P R E S E N T A :

GERARDO LUNA VENTURA

DICTAMINADORES

DR. JORGE ENRIQUE WALS SELVAS

DR. JOSIANE JAIME RODRÍGUEZ SUÁREZ

Ciudad de México, noviembre de 2023.

.

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Agradecimientos

Estoy profundamente agradecido con Dios por haberme permitido llegar hasta donde estoy hoy en día. También agradezco que siempre me ha brindado todo lo necesario para lograr mis objetivos. Sin su ayuda y bendiciones, nada de esto habría sido posible.

A mis padres, que con todo el esfuerzo y dedicación me dieron todo lo necesario para sacar adelante mis estudios, sin importar los tiempos difíciles que pasamos, también en muchas ocasiones no supe ni cómo le hacían para seguir adelante, pero siempre estaban ahí apoyándome.

A mis hermanos y familiares que siempre han estado presentes cuando los necesito. Siempre me han apoyado en lo que han podido, además de ser un soporte que me impulsa a seguir adelante.

A mi novia y fiel compañera Alondra, que siempre ha estado conmigo en las buenas y en las malas, apoyándome siempre en mis proyectos.

Y a todos mis profesores de la UACM. Durante años, esta institución fue mi segunda casa, gracias a ellos hoy en día cuento con un gran bagaje de conocimientos y herramientas que me han preparado para enfrentar el mundo laboral. Su dedicación y compromiso con mi formación son invaluables, siempre estaré agradecido por su valiosa contribución a mi desarrollo profesional.

Resumen

El objetivo de este documento es demostrar los conocimientos adquiridos a lo largo de la Licenciatura en Ingeniería de Software, aplicados en el ámbito laboral y enfocado específicamente en el diseño de casos de prueba en los proyectos de desarrollo de software en los que he participado. En este sentido, presento algunas definiciones y conceptos relevantes que un *tester* debe conocer.

También expongo algunos datos relevantes, que pueden ser de gran ayuda para mis compañeros de la Licenciatura en Ingeniería de Software, que quieren tomar el camino para convertirse en *testers* y algunos consejos importantes que les ayudarán en esta especialidad, que hasta el día de hoy sigue creciendo en cuanto a la demanda laboral.

Además, expongo las técnicas de pruebas de software que más he utilizado en los proyectos donde he laborado, con ejemplos prácticos para ver cómo se utilizan y aplican a la hora de diseñar casos de prueba. Cabe señalar que, los ejemplos presentados son experiencias aplicadas en 3 años laborados para una empresa enfocada en el desarrollo de software, en el área de *testing*.

Al final, presento una conclusión acerca de cómo las técnicas de pruebas de software me han permitido diseñar mejores casos de prueba y una perspectiva personal de cuándo aplicar pruebas manuales o pruebas automatizadas según sea el caso.

TABLA DE CONTENIDO

LISTA DE IMÁGENES

LISTA DE TABLAS

Capítulo 1: Conceptos importantes que debe conocer un *Tester*

1.1. ¿Qué son las pruebas de software?

El *International Software Testing Qualifications Board* ISTQB (2018) indica que "las pruebas de software son una forma de evaluar la calidad del software y para reducir el riesgo de fallas en el funcionamiento". (p. 13)

También el *Institute of Electrical and Electronics Engineers* IEEE (1990) nos dice que las pruebas se definen como "el proceso de analizar un elemento de software para detectar las diferencias entre las condiciones existentes y requeridas (es decir, errores) y para evaluar las características de los elementos de software". (p. 76)

De las dos definiciones antes citadas, podemos entender que las pruebas de software son una forma de evaluar la calidad del software, donde la calidad es precisamente que cumpla con lo establecido en los requerimientos y así evitar discrepancias o fallas en el funcionamiento. También se entiende que, las pruebas de software son parte importante del desarrollo, y el resultado de realizar buenas pruebas es la entrega de un producto de calidad, permitiendo que el nuevo software sea una herramienta importante para quien lo solicita.

1.2. ¿Qué es el Aseguramiento de la Calidad de Software?

Acerca de las definiciones del aseguramiento de la calidad de software el *International Software Testing Qualifications Board* ISTQB (2018) menciona lo siguiente:

El aseguramiento de la calidad y las pruebas no son iguales, pero están relacionadas. Un concepto más amplio, la gestión de la calidad, los une. La gestión de la calidad incluye todas las actividades que dirigen y controlan una organización con respecto a la calidad. Entre otras actividades, la gestión de la calidad incluye tanto la garantía de calidad como el control de calidad. El aseguramiento de la calidad generalmente se enfoca en la adherencia a los procesos adecuados, con el fin de brindar confianza en que se lograrán los niveles adecuados de calidad. Cuando los procesos se llevan a cabo correctamente, los productos de trabajo

creados por esos procesos son generalmente de mayor calidad, lo que contribuye a la prevención de defectos. Además, el uso del análisis de la causa raíz para detectar y eliminar las causas de los defectos, junto con la aplicación adecuada de los resultados de las reuniones retrospectivas para mejorar los procesos, son importantes para garantizar una calidad eficaz. (p. 15)

También para complementar la definición acerca del aseguramiento de la calidad de software el *Institute of Electrical and Electronics Engineers* IEEE (1990) explica lo siguiente:

> Un patrón planificado y sistemático de todas las acciones necesarias para brindar la confianza adecuada de que un artículo o producto se ajusta a los requisitos técnicos establecidos. (p. 60)

El aseguramiento de calidad (QA, por sus siglas en inglés), es una actividad fundamental para validar que todos los estándares, reglas y procesos de calidad aplicables al nuevo producto (en este caso, software) se cumplan. El objetivo de esta actividad es asegurar que la calidad del producto sea consecuencia de la implementación adecuada de estos procesos, lo que le brinda madurez, seguridad y robustez para competir en el mercado. En este sentido, el aseguramiento de calidad es fundamental para que el software se convierta en un activo importante para la empresa o institución que lo ha desarrollado, volviéndose una solución confiable y efectiva para satisfacer sus necesidades.

Es importante mencionar que, si en todos los procesos a lo largo de todo el ciclo de vida de desarrollo de software, siempre está presente el aseguramiento de la calidad y se siguen estándares alcanzando cierta madurez, las pruebas de software serán más rápidas, precisas y de calidad como todo lo demás. En cambio, sí desde un principio el desarrollo del nuevo software no se realiza con la mayor calidad posible, todas las etapas subsecuentes tendrán problemas de calidad, complicando la etapa de pruebas.

Hoy en día, existen modelos que revisan el cumplimiento de procesos, estándares y tareas de calidad por parte de las organizaciones, con el fin de determinar el grado de madurez del desarrollo en curso. Estos modelos también evalúan el nivel de madurez alcanzado por la institución, permitiendo avanzar gradualmente en cada ciclo y mejorar la calidad del

desarrollo de software. Un ejemplo destacado de estos modelos es el *Capability Maturity Model Integration* (CMMI, por sus siglas en inglés).

1.3. Antecedentes de las pruebas de software

Las pruebas en general son realizadas en todos los proyectos de cualquier tipo que implique el desarrollo de algún producto o servicio, permitiendo que este cuente con la calidad requerida o sirva para lo que fue desarrollado. Es por esto y más que, en el desarrollo de software como cualquier otro tipo de producto se llevan a cabo pruebas de calidad, que permitan garantizar el correcto funcionamiento de este y que además permitan identificar errores o mejoras dentro del software, antes de que estos errores impacten de lleno a la operación o generen riesgos graves.

Algunos ejemplos reales de la consecuencia de no realizar pruebas de software correctas según Hamilton (2023), se listan a continuación:

> En abril de 2015, la terminal de Bloomberg en Londres colapsó debido a una falla de software que afectó a más de 300,000 comerciantes en los mercados financieros. Obligó al gobierno a posponer una venta de deuda de 3.000 millones de libras.
>
> Los automóviles Nissan retiraron del mercado más de 1 millón de automóviles debido a una falla del software en los detectores sensoriales de las bolsas de aire. Se han reportado dos accidentes debido a esta falla del software.
>
> Starbucks se vio obligado a cerrar alrededor del 60 por ciento de las tiendas en los EE. UU. y Canadá debido a una falla de software en su sistema POS. En un momento, la tienda sirvió café gratis porque no pudieron procesar la transacción.
>
> Algunos de los minoristas externos de Amazon vieron que el precio de su producto se redujo a 1p debido a una falla de software. Se quedaron con grandes pérdidas.
>
> Vulnerabilidad en Windows 10. Este error permite a los usuarios escapar de los entornos limitados de seguridad a través de una falla en el sistema win32k.
>
> En 2015, el avión de combate F-35 fue víctima de un error de software que le impidió detectar objetivos correctamente.

El Airbus A300 de China Airlines se estrelló debido a un error de software el 26 de abril de 1994, matando a 264 inocentes en vivo.

En 1985, la máquina de radioterapia Therac-25 de Canadá no funcionó correctamente debido a un error de software y entregó dosis letales de radiación a los pacientes, lo que provocó la muerte de 3 personas y heridas graves a otras 3.

En abril de 1999, un error de software provocó el fracaso del lanzamiento de un satélite militar de 1200 millones de dólares, el accidente más costoso de la historia.

En mayo de 1996, un error de software provocó que las cuentas bancarias de 823 clientes de un importante banco estadounidense se acreditaran con 920 millones de dólares estadounidenses.

La lista anterior nos dice mucho sobre el impacto de no realizar pruebas de software correctamente y las consecuencias (algunas hasta han cobrado vidas humanas) de realizar pruebas sin cubrir bien la mayoría de los escenarios posibles, (más adelante se expone una técnica de pruebas de software que ayuda a cubrir la mayoría de los escenarios posibles a la hora de diseñar pruebas) en los cuales viven defectos que ponen en riesgo la operación del software.

Las pruebas de software han experimentado una notable evolución, brindando actualmente una amplia gama de técnicas, metodologías y herramientas que facilitan este trabajo, agilizando el diseño y ejecución. Es importante destacar que esto no siempre fue así, ya que en el pasado las pruebas de software carecían de importancia y solo se realizaban en una etapa específica del ciclo de vida del desarrollo de software, sin profundizar demasiado en los requerimientos. Su enfoque principal consistía en certificar que el software funcionara correctamente para luego pasar a la fase de operación. Sin embargo, en la actualidad, se reconoce la relevancia de las pruebas de software desde las etapas iniciales del desarrollo, con un enfoque más integral y exhaustivo en los requerimientos y en la detección temprana de errores. También, las pruebas de software han presentado varios cambios importantes, en la manera de cómo se realizan, permitiendo mejoras para estar a la altura de los proyectos que cada día son más complejos o que requieren de mayor esfuerzo por parte del *tester*.

Comprender la necesidad de realizar pruebas en el desarrollo de software es de suma importancia y resulta esencial conocer las consecuencias de no realizar pruebas o de no llevar

a cabo las pruebas adecuadas, ya que en numerosas ocasiones a lo largo del tiempo esto ha puesto en riesgo la reputación e incluso la propia operación de instituciones importantes. Por tanto, aprender de estas experiencias nos permite apreciar la relevancia y el impacto de las pruebas de software en la calidad y el éxito de los sistemas y aplicaciones.

1.4. Tareas que lleva a cabo el *Tester*

El *tester* es la persona encargada de velar por la calidad del software, utilizando todas las herramientas, técnicas y metodologías que existen para llevar a cabo su trabajo.

Los *testers* a menudo tienen que trabajar a lo largo de todo el ciclo de vida de desarrollo del software, interactuando con las diferentes áreas para recabar la mayor cantidad de información que le permitan diseñar pruebas de calidad, cubriendo todos los escenarios posibles, minimizando la cantidad de defectos presentes en el nuevo software. También los *testers* tienen que realizar un análisis exhaustivo desde la etapa del desarrollo de software en la que se involucran, de esta manera pueden detectar defectos en etapas tempranas del proyecto, impactando de manera positiva, ya que estos tienen un menor costo para solucionarse a diferencia de que si se detectan en etapas avanzadas.

Además, el *tester* es el responsable de reportar a los interesados del proyecto los resultados de las diferentes pruebas realizadas, para así determinar si el nuevo producto de software está cumpliendo con lo especificado en los requerimientos y si es apto para la operación.

Un ejemplo de la importancia del trabajo del *tester* es el siguiente:

Cuando se realiza una nueva entrega de software al área de Control de Calidad (QA) para llevar a cabo las pruebas correspondientes, especialmente en el primer ciclo de pruebas, los resultados obtenidos pueden tener un impacto significativo en todo el proyecto de desarrollo de software. En muchas ocasiones, tras generar el primer informe de pruebas y si los resultados no son favorables (con fallas en la mayoría de las pruebas), se suele devolver la entrega inicial al área de desarrollo y se propone una nueva fecha para reanudar las pruebas. Esto se debe a que los defectos presentes en el nuevo software son numerosos, y no vale la pena invertir esfuerzos en probar un software que no cumple con la mayoría de los requerimientos. Es por esto y más que el trabajo del *tester* es muy importante, siempre se debe realizar con mucho empeño, siendo minucioso a la hora de probar software.

1.5. Algunos riesgos a los que se enfrenta el *Tester*

A lo largo de un proyecto de desarrollo de software, por lo general el *tester* tiene que enfrentarse a varios riesgos, que le impiden o le complican la manera de llevar a cabo su trabajo.

Algunos riesgos se enlistan a continuación:

- El producto de software presenta cambios que no se informaron al área de QA por lo que las pruebas no cubrirán los nuevos requerimientos.
- El producto de software no se libera a tiempo al área de QA para que se inicien las pruebas pertinentes.
- Hay cambios constantes en los requerimientos de software y el área de QA tiene que estar realizando esfuerzos constantes para actualizar las pruebas a realizar.
- No hay documentación para dar soporte a las pruebas.
- Los tiempos estimados para las pruebas no son realistas.
- Los usuarios se niegan a apoyar las pruebas.
- No existe comunicación en el equipo de pruebas.
- No hay herramientas de apoyo a las pruebas.
- No hay credenciales o accesos al software para los *testers,* o estas tardan demasiado en otorgarse.
- El presupuesto no es suficiente para cubrir las pruebas correctamente.
- No se da suficiente importancia a las pruebas y solo se realizan pruebas de caja negra.
- Los *testers* no están realizando las pruebas correctas.
- La comunicación con los desarrolladores es mínima o no existe.
- El área de desarrollo no resuelve los defectos reportados.
- El área de desarrollo no resuelve los defectos reportados, como bloqueantes con la urgencia requerida.
- El área de desarrollo no comprende la descripción de los defectos y no busca apoyo con el área de QA para su atención.
- La atención de los defectos es muy lenta.

Además de los riesgos antes listados, existen problemas ajenos, los cuales impactan constantemente las pruebas y pone en riesgo la calidad del producto. Un ejemplo muy común son los cambios constantes en el software, donde se agregan nuevas funcionalidades que antes no estaban planeadas. El *tester* destina mucho tiempo en diseñar y ejecutar las pruebas, pero si en ese lapso se solicita algún cambio, se tendrá que aplicar mayor esfuerzo para volver a iniciar el ciclo de pruebas y cubrir estos cambios imprevistos.

1.6. Mitigar los riesgos a los que se enfrenta el *Tester*

A lo largo de todo el ciclo de vida del desarrollo de software, el *tester* debe mitigar los riesgos a los que se enfrenta, esto con el fin de lograr la calidad requerida para el nuevo software.

Dentro de la documentación, en la planeación, se deben exponer los principales riesgos a los que se puede enfrentar el equipo de pruebas y la manera en que se deben mitigar estos riesgos.

Los riesgos que se van presentando a lo largo del ciclo de pruebas y que no están documentados, se deben exponer en las reuniones para que todo el equipo acuerde soluciones efectivas que permitan avanzar, minimizando el impacto a todo el proyecto. En estas reuniones se acuerdan la mayoría de las soluciones a los problemas no previstos. Dependiendo de la metodología de desarrollo que se implemente, se deben tener reuniones constantes para así lograr una comunicación efectiva tanto en el equipo de pruebas como con todo el equipo del proyecto en general.

Siempre es importante la comunicación con el equipo de pruebas y con todo el equipo de desarrollo, para trabajar en armonía y con objetivos comunes.

A continuación, se muestra una lista con los ejemplos de mitigación de riesgos más comunes que enfrenta el *tester*, mencionadas en la sección anterior:

Riesgo	**Acción por realizar**
El producto de software presenta cambios que no se informaron al área de QA, por lo que las pruebas no cubrirán los nuevos requerimientos.	Se debe informar por escrito que, una vez iniciadas las pruebas, cualquier cambio en los requerimientos debe ser informado inmediatamente al área de QA para su respectiva atención y seguimiento. Otra acción sería que el cambio se documente como nuevo requerimiento o

	mejora, para ser atendido en entregas posteriores.
El producto de software no se libera a tiempo al área de QA para que se inicien las pruebas pertinentes.	Se debe pactar una fecha de entrega del nuevo software al área de QA para comenzar las pruebas necesarias a la brevedad posible. Si la fecha se modifica, se debe ajustar todo el plan de pruebas (en caso de ser necesario).
Hay cambios constantes en los requerimientos de software y el área de QA tiene que estar realizando esfuerzos constantes en actualizar las pruebas a realizar.	Se debe pactar una fecha en la cual no se atenderán cambios adicionales hasta que se concluya el ciclo de pruebas en el que se está trabajando.
No hay documentación para dar soporte a las pruebas.	Se debe recabar la información necesaria para realizar las pruebas de donde se pueda, ya que es necesario tener la mayor cantidad de información posible.
Los tiempos estimados para las pruebas no son realistas.	Se debe realizar una estimación del esfuerzo de pruebas, dependiendo de la magnitud del proyecto y del tipo de pruebas a realizar. También influyen mucho los recursos que se tengan destinados para las pruebas y las herramientas disponibles.
Los usuarios se niegan a apoyar las pruebas.	Se debe acordar con los usuarios fechas y tiempos destinados al apoyo de las pruebas. Es importante la participación del usuario final para dar visto bueno al nuevo software y también corregir lo que se considere necesario en las pruebas de aceptación de usuario.
No existe comunicación en el equipo de pruebas.	Se deben acordar reuniones periódicas para aclarar cualquier tema relacionado con el proyecto, así como también se deben tener diferentes canales de comunicación, (Email, herramientas de mensajería instantánea, herramientas de colaboración, etc.) que permitan que tanto el equipo de pruebas como todo el equipo del proyecto en general, tenga comunicación efectiva todo el tiempo.
No hay herramientas de apoyo a las pruebas.	Se deben gestionar herramientas de trabajo al momento de iniciar el proyecto de pruebas, ya que son necesarias para facilitar el seguimiento y ejecución de estas. Existen

	diferentes herramientas de apoyo a las pruebas de uso libre y de paga que facilitan el trabajo del *tester*.
No hay credenciales o accesos al software para los *testers*, o estas tardan demasiado en otorgarse.	Se deben solicitar accesos al área encargada en fechas tempranas a la entrega del software, para que el área de pruebas pueda iniciar la revisión en tiempo y forma. También se debe indicar que la prioridad es alta, puesto que, de no atenderse la petición, podría generar un retraso en las pruebas, impactando a todo el proyecto en general.
El presupuesto no es suficiente para cubrir las pruebas correctamente.	El equipo de pruebas debe apegarse lo más posible al presupuesto, pero si el presupuesto no es realista, se debe levantar la mano para gestionar más presupuesto y llegar a un acuerdo que permita tener los recursos necesarios para realizar las pruebas correctas. De no llegar a un acuerdo satisfactorio, se debe informar por escrito que existe un riesgo, donde las pruebas pueden no ser suficientes y así deslindar responsabilidades para el *tester.*
No se da suficiente importancia a las pruebas y solo se realizan pruebas de caja negra.	Se debe exponer en las reuniones con todo el equipo del proyecto que, las pruebas deben tener un gran peso y que deben ejecutarse varios tipos de pruebas, (Caja negra, caja blanca, funcionalidad, carga/estrés, seguridad, etc.) que aseguren la calidad del nuevo software.
Los *testers* no están realizando las pruebas correctas.	Desde el inicio del ciclo de pruebas, los *testers* deben analizar los tipos de pruebas necesarios para asegurar la calidad del software y deben poner énfasis en algún tipo de pruebas, dependiendo del tipo de software que se esté desarrollando. Ejemplo: Si se trata de una aplicación bancaria, las pruebas de seguridad deben tener prioridad alta.
La comunicación con los desarrolladores es mínima o no existe.	Se deben tener canales de comunicación directos con los desarrolladores, así como también se deben tener reuniones periódicas, ya sea en conjunto o uno a uno, para dar seguimiento a dudas y temas que afecten el trabajo del *tester* o del desarrollador.

El área de desarrollo no resuelve los defectos reportados.	Se debe exponer en las reuniones con el equipo de desarrollo, la importancia de dar solución a los defectos que impiden que el trabajo del *tester* avance correctamente. En caso de que los temas tratados no sean atendidos, se deben escalar con el siguiente nivel (con el líder de proyecto o los interesados del proyecto) para que sean atendidas las peticiones indicadas. Es importante tener documentado o registrado el seguimiento y las respuestas por parte del equipo de desarrollo, para así tener evidencia del seguimiento que se les dé a los defectos levantados por el equipo de pruebas.
El área de desarrollo no resuelve los defectos reportados, como bloqueantes con la urgencia requerida.	Este riesgo se debe mitigar igual que el riesgo anterior pero adicionalmente se debe indicar en las reuniones, correos y documentos que los defectos son bloqueantes y que de no ser atendidos, el impacto será mayor que el de otros defectos, poniendo en riesgo el proyecto en general. Con esto se busca que los interesados del proyecto estén enterados de la gravedad del asunto y tomen las medidas necesarias.
El área de desarrollo no comprende la descripción de los defectos y no busca apoyo con el área de QA para su atención.	El equipo de pruebas debe indicar en las reuniones, documentos y correos que, siempre están abiertos a atender cualquier tema relacionado con los defectos, permitiendo la comunicación efectiva, en todo momento.
La atención de los defectos es muy lenta.	Se debe solicitar al área de desarrollo la atención de los defectos lo más pronto posible. Es importante recordarles que en caso de que tengan algún tema que no permita que los defectos sean atendidos, esto sea notificado por escrito y escalado hasta los niveles más altos, (en caso de ser necesario) para que se tomen las acciones pertinentes.

Tabla 1: Lista de mitigación de algunos riesgos

Capítulo 2: De estudiante a *Tester*

2.1. ¿Por qué decidí especializarme como *Tester*?

En esta sección expongo los principales motivos por los que decidí especializarme como *tester*, enfocado en el aseguramiento de la calidad del software. Estos motivos son basados en las experiencias vividas, acontecidas durante y después de la universidad.

1. En la Universidad Autónoma de la Ciudad de México, se ofrece la Licenciatura en Ingeniería de Software, la cual incluye unidades de aprendizaje como "Aseguramiento de la calidad de software" y "Técnicas de pruebas de software". Estas asignaturas son impartidas por docentes especializados que brindan conocimientos sobre el aseguramiento de la calidad del software, así como diversas técnicas de prueba que se pueden utilizar para diseñar casos de prueba adaptados a los requerimientos específicos. Durante mi formación académica dentro de la UACM, tuve mi primer acercamiento a las técnicas de pruebas de software, lo que me permitió comprender cómo se llevan a cabo las pruebas en los proyectos de desarrollo de software y me gustó mucho. Sin embargo, no sabía entonces que estas técnicas serían una herramienta fundamental para mi carrera profesional. Gracias a los conocimientos adquiridos en mi casa de estudios, pude enfrentar el mundo laboral con confianza y demostrar mi valía, lo que me permitió destacar y poner en alto el nombre de mi universidad.

2. Desde la perspectiva de un usuario común que utiliza sistemas software, ya sea en web o en dispositivos móviles, siempre he enfrentado problemas con sistemas que no cumplen con los estándares de calidad necesarios para que un usuario pueda trabajar adecuadamente. Los sistemas que más dificultades me han generado son aquellos relacionados con registros, ya sea en el ámbito gubernamental o en plataformas bancarias. Estos sistemas presentan constantes problemas al cargar las páginas y, en ocasiones, después de completar todo el proceso de ingreso de datos, simplemente se caen o no guardan los registros correctamente. Además, considero que el hecho de que un software consuma recursos excesivos de los equipos de cómputo revela una falta de

calidad, lo cual genera malas experiencias para el usuario y afecta negativamente el rendimiento en las tareas que se están realizando.

3. Siempre me ha gustado ser crítico de los sistemas software, con expectativas muy altas. Esto me da pie a preguntarme, ¿cómo se podrían mejorar los sistemas? y ¿cómo se pueden hacer mejores sistemas de software?, para que los usuarios se lleven buenas experiencias en el uso de estos.

4. En mi primer trabajo enfocado en tecnologías de la información, comencé como auxiliar de soporte técnico para un sistema financiero nuevo, de una empresa bancaria. Durante esta experiencia me di cuenta de que el sistema carecía de calidad, y lo confirmé cuando contrataron a un equipo de más de 100 personas (incluyéndome a mi) solo para levantar reportes de fallos, los cuales eran constantes. Además, los usuarios en las sucursales informaban que varios clientes se marchaban, debido a los problemas que impedían la contratación de los productos que requerían, lo que representaba pérdidas notables para el negocio. Todo esto me permitió comprender la importancia de garantizar la calidad del software, ya que los fallos pueden afectar tanto a la experiencia del usuario como la rentabilidad del negocio. También comprendí que, si un sistema software no cumple con los buenos criterios de calidad desde el principio, más adelante puede ser muy caro realizar el mantenimiento correctivo, por lo que las pruebas tempranas y correctas disminuyen mucho los costos y los riesgos.

Después de haber cursado las asignaturas de Aseguramiento de la calidad de software y Técnicas de pruebas de software en mi universidad, así como haber tenido mi primera experiencia laboral con un sistema deficiente en calidad, pude percatarme de mi atracción por probar el funcionamiento de los sistemas software. Además, al investigar sobre las oportunidades laborales en el campo de tecnologías de la información, me di cuenta de que numerosos sitios web, revistas y periódicos mencionaban la creciente demanda de profesionales especializados en aseguramiento de la calidad para el desarrollo de software, siguiendo las últimas metodologías. Teniendo en cuenta todos estos aspectos, decidí enfocar mi especialización a *tester*, dedicado específicamente al aseguramiento de la calidad de los sistemas software.

2.2. Procesos de reclutamiento en mi primer empleo como *Tester*

En los procesos de reclutamiento para trabajar como *tester*, actualmente toman en cuenta el conocimiento sobre las pruebas de software y cómo dar solución a los problemas, específicamente diseñando y ejecutando casos de prueba.

Durante mi primer empleo como *tester*, fui evaluado por un especialista en pruebas, con el objetivo de conocer mis habilidades. En esta experiencia tuve la responsabilidad de diseñar y ejecutar casos de prueba para un sistema determinado. También evaluó mi conocimiento en programación y bases de datos (lenguajes de programación y manejadores de bases de datos que conocía).

En el proceso de reclutamiento de mi primer empleo como *tester* me encontré con varios compañeros de distintas universidades, de los cuales muchos tenían noción de cómo aplicar las técnicas de pruebas de software, pero muchos desconocían el tema, por lo cual concluí que los egresados de mi universidad cuentan con los conocimientos necesarios para ejercer su profesión como cualquier otro egresado de las demás universidades y los límites solo están en nuestras mentes.

Esta experiencia me permitió adquirir habilidades valiosas en pruebas de software, así como mejorar mis conocimientos en programación y bases de datos.

2.3 Consejos para los compañeros que quieran ser *Testers*

A los compañeros estudiantes de mi casa de estudios que busquen incorporarse al mundo laboral con el enfoque en el aseguramiento de la calidad de software como *testers*, les diría que:

1. Busquen mejorar sus habilidades de comunicación con las demás personas y con sus equipos de trabajo. Es importante la comunicación constante en todo momento debido a que, al no haber comunicación, se puede ver afectado el éxito del proyecto en el que trabajen. Además, como *tester* es importante que no se tenga duda alguna de como probar un sistema, ya que, si no se busca entender bien el funcionamiento de lo que se pretende probar, las pruebas no serán las correctas o no se realizarán correctamente.

2. Es importante elevar su nivel inglés, por lo menos hasta un nivel intermedio, tanto para hablar como para escribir, ya que, es necesario estar en constante comunicación con personas de habla inglesa y la mayor parte de la información necesaria para llevar a cabo su trabajo está en inglés. Además, los mejores empleos para los *testers* están en las empresas donde requieren un nivel de inglés intermedio - avanzado, para tener comunicación con personas de diferentes países.

3. Siempre estar a la par con los cambios que hay en esta área, ya que constantemente hay nuevas técnicas, herramientas, metodologías y plataformas que facilitan o mejoran el trabajo del *tester*. Además, en los empleos constantemente te piden conocimientos en el manejo de estas técnicas, herramientas, metodologías y plataformas con sus respectivas actualizaciones.

4. Especializarse en algún tipo de pruebas es muy importante ya que permite llegar directamente a realizar pruebas que se conocen y/o manejan muy bien, lo cual es un punto a favor a la hora de buscar empleo. Estas pruebas pueden ser funcionales, no funcionales, de seguridad, pruebas unitarias, etc.

5. Deben certificarse como *testers* y en el uso de distintas herramientas y/o plataformas. Es importante que un *tester* cuente con certificaciones ya que los empleadores constantemente buscan personal certificado a la hora de seleccionar nuevos candidatos. Además, cuando se realiza la certificación se adquieren nuevos conocimientos, los cuales permiten avanzar en esta especialidad. Hay muchas instituciones donde pueden certificarse, estas pueden ser nacionales e internacionales pero la que más les recomiendo es ISTQB, donde hay un nivel básico (*Foundation Level*) y sus diferentes especialidades (*Advanced Level*).

6. Deben conocer o especializarse en algún lenguaje de programación que les permita entender código, ya que actualmente los empleos requieren personal que entienda bien la lógica de programación para realizar pruebas *FrontEnd* y *BackEnd.* Además, se

requieren conocimientos en programación para realizar *scripts* de pruebas automatizadas. Si saben hacer pruebas automatizadas, sus opciones de empleos se duplican ya que los empleos con mejores percepciones económicas tienen como requisito saber algún lenguaje de programación para automatizar o actualizar una automatización.

7. Para buscar empleo como *tester* pueden buscar en redes sociales, plataformas de empleos o páginas especializadas y verán las ofertas de empleos que existen actualmente con los requerimientos que solicitan para un *tester*. También hay grupos de *testers* en las diferentes redes sociales, donde constantemente publican cursos gratuitos de herramientas y/o de especialidades que son de suma importancia.

2.4 Demanda de empleo para *testers* en el 2023

Un punto muy importante que he decidido mencionar en este reporte de titulación es la demanda de empleo para *testers* en este año 2023, ya que es información muy relevante para los compañeros que quieran volverse especialistas en pruebas y tengan dudas de si es algo que va creciendo en cuanto a la demanda laboral. Cabe señalar que la información aquí presentada, es solo con fines informativos y no hay garantía de que la demanda de empleo sea creciente siempre, pero en el año actual en que se realizó este reporte si es creciente según varios sitios especializados.

A continuación, la siguiente imagen lista algunos perfiles de TI más cotizados y con más demanda de empleo en el 2022.

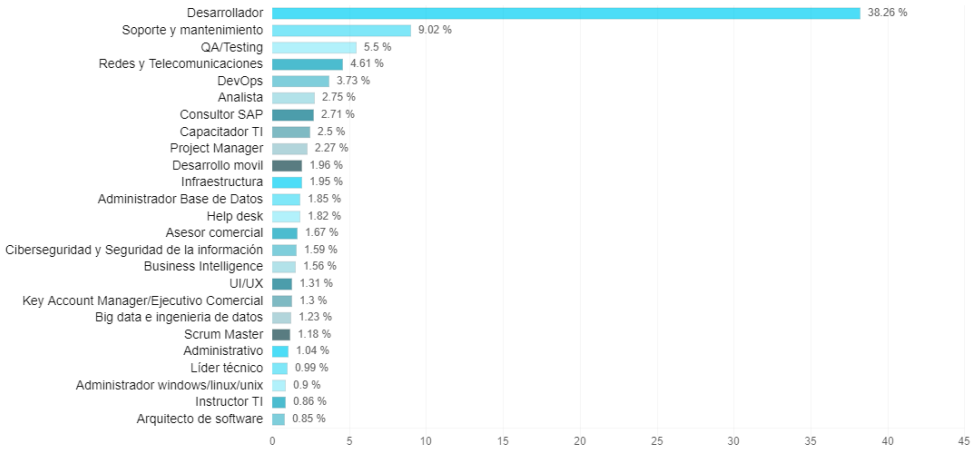


Imagen 1: Demanda de empleo para TI según el Reporte del Mercado Laboral de TI en LATAM 2023 de Hireline. Fuente Olvera (s.f.)

En cuanto a la demanda laboral para testers, Olvera (s.f.) en su Reporte del Mercado Laboral de TI en LATAM 2023 menciona que, el perfil de QA/Testing se encuentra en tercer lugar de los perfiles más demandados en Latinoamérica con un 5.5% del total de las vacantes de TI disponibles analizadas para el reporte.

Para ver el reporte completo antes mencionado se puede acceder desde el siguiente enlace o *link*: https://hireline.io/remoto/estudio-mercado-laboral-y-empleos-de-ti-latam?year=2023

En base a lo analizado en la imagen de arriba, se observa que la especialidad de QA Tester puede ser muy buena opción para encontrar empleo rápidamente y con una remuneración económica mucho mejor que otros perfiles de TI para el año 2023.

Capítulo 3: Experiencias en diseño de pruebas de software

3.1. Experiencias en diseño de pruebas

A continuación, una pequeña reseña de la experiencia que he tenido en el diseño de pruebas para un proyecto de desarrollo de software, en una de las empresas donde he laborado.

PRUEBAS EN UN PROYECTO DE SEGURO DE GASTOS MÉDICOS MAYORES COLECTIVOS E INDIVIDUALES

En el proyecto en el que participé como tester manual, el enfoque inicial del equipo de pruebas era comprender el objetivo global del proyecto. Posteriormente, nos adentrábamos en la comprensión de los requerimientos, que podían ser tanto historias de usuario como fichas técnicas o requisitos específicos. Inicialmente, nos encontramos con una falta de documentación suficiente para el diseño de los casos de prueba. Por lo tanto, recurrimos a las reuniones de toma de requerimientos, donde nos esforzábamos por comprender las necesidades de los usuarios. A partir de este entendimiento, comenzábamos a desarrollar un diseño preliminar de casos de prueba, complementado con fichas técnicas e historias de usuario. Era fundamental para el *Test Manager* recibir un entregable inicial que demostrara el progreso de nuestro trabajo. Sin embargo, estos entregables carecían de madurez, ya que los casos de prueba aún no estaban lo suficientemente desarrollados para proceder con la ejecución. Además, nos faltaban los datos necesarios para llevar a cabo las pruebas de manera efectiva.

En el diseño de los casos de prueba solicitamos información a los interesados del proyecto, y si detectábamos algún requerimiento que no era claro para nosotros, se acordaba juntas con los usuarios finales para apoyarnos en ellos y así entender mejor lo que se esperaba del nuevo software. Estas reuniones eran específicamente entre el usuario final y el equipo de pruebas para así afinar detalles, permitiendo que el diseño de los casos de prueba fuera correcto. Así mismo, el usuario final se encargaba de revisar los casos de prueba, con la finalidad de encontrar algún error o contradicción con los requerimientos.

Después de creer que habíamos entendido los requerimientos de manera correcta (o eso pensábamos en ese momento), procedíamos al diseño de los casos de prueba utilizando la herramienta de Excel en línea. Los casos de prueba se elaboraban y redactaban en un solo documento que estaba sujeto a revisión y supervisión por parte de cualquier persona interesada en el proyecto. Asimismo, para el diseño de los casos de prueba, se asignaban módulos del software a cada miembro del equipo de pruebas, lo que implicaba que cada uno era responsable del diseño y ejecución de los casos de prueba correspondientes a su módulo asignado. Además, los casos de prueba diseñados por cada miembro del equipo eran revisados por otro *tester*, con el objetivo de asegurar que fueran comprensibles para cualquier miembro del equipo y que cumplieran con lo solicitado en cada requerimiento cubierto.

Al finalizar la fase de diseño de los casos de prueba del primer alcance, se procedía a la ejecución de las pruebas. Durante esta etapa, se registraban y reportaban los defectos encontrados, llevando a cabo un seguimiento de estos hasta su resolución. Es crucial subrayar que la responsabilidad de dar seguimiento a los defectos y obtener la aprobación del usuario recaía siempre en el *tester*. Asimismo, disponíamos de documentación que respaldaba el reporte, seguimiento y cierre de los defectos, lo cual aseguraba una correcta solución de los problemas identificados.

En conclusión, durante mi participación en el proyecto de desarrollo del software de seguros de gastos médicos mayores, pude observar que en ocasiones los requerimientos no se recopilan de manera precisa. Esto suele ser resultado de la falta de información o la omisión de detalles importantes en el proceso de toma de requerimientos. En algunos casos, incluso los desarrolladores pueden tener dificultades para comprender las tareas a realizar y se limitan a programar lo que pueden. Esto conduce a pruebas inadecuadas que no cumplen con las expectativas del usuario, ya que se centran únicamente en cumplir con lo solicitado en lugar de abordar las necesidades reales. Como consecuencia, esto puede afectar la satisfacción del usuario final y la calidad del software entregado. Para mitigar este riesgo, es crucial que el *tester* colabore estrechamente con los interesados del proyecto o los usuarios finales para recopilar la información necesaria y así llevar a cabo pruebas correctas.

Actualmente existen muchas herramientas que posibilitan el seguimiento de los requerimientos y la gestión de defectos, facilitando una trazabilidad desde su identificación hasta su resolución. Estas herramientas permiten que todos los involucrados en el proyecto puedan supervisar el progreso de las tareas, incluyendo historias de usuario, requerimientos, épicos y defectos. Esto se logra gracias a que las métricas se generan de manera automática, brindando una visión completa del avance.

3.2. Diseño de pruebas funcionales

En los 3 años laborados en el área de QA mi enfoque se limitó en realizar solo pruebas funcionales de caja negra, por lo que el diseño de los casos de prueba que realicé siempre fue enfocado en los entregables por parte del área de desarrollo. Las pruebas que realicé eran específicamente en probar los requerimientos, donde el diseño de los casos de prueba era enfocado en el funcionamiento de los sistemas y avanzaba a medida que el área de desarrollo

liberaba módulos o funcionalidades. Además, las pruebas realizadas se apegaban a las metodologías ágiles en las que se trabajaba.

Es importante destacar que, durante todo el proceso de pruebas, siempre me enfoqué en detectar cualquier discrepancia que existiera entre los requerimientos y el funcionamiento real del sistema. Para lograr esto, cada caso de prueba debía especificar claramente el requerimiento al cual estaba apuntando, lo cual permitía establecer una trazabilidad adecuada entre los requerimientos y los casos de prueba. Esta trazabilidad resulta esencial para identificar y solucionar los problemas detectados en el proceso de pruebas, asegurando que se cumplen todas las necesidades del usuario y los objetivos del proyecto.

Capítulo 4: Experiencias con técnicas de pruebas de software

4.1. ¿Qué es un caso de prueba?

La definición de caso de prueba según el *Institute of Electrical and Electronics Engineers* IEEE (1990) se expone a continuación:

> Un conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados desarrollados con un objetivo particular, tal como el de ejercitar un camino en particular de un programa o el verificar que cumple con un requerimiento específico. (p. 74)

Otra definición de caso de prueba es la que explica Toledo (2014) y que también se expone a continuación:

> Un caso de prueba específico (o concreto) es una instancia de un caso de prueba abstracto, en la que se determinan valores específicos para cada variable de entrada y para cada salida esperada. (p. 28)

Dadas las definiciones anteriores comprendo que, un caso de prueba se puede definir como un escenario específico en el cual se tiene un conjunto de entradas, condiciones, pasos a seguir y resultados esperados que especifican los resultados que debe cumplir cierta funcionalidad del software a probar, en las condiciones especificadas. Además, tiene como objetivo recabar

evidencia que demuestre el funcionamiento del software para su posterior consulta y/o seguimiento.

4.3. ¿Qué es una técnica de pruebas de software?

Según el *Institute of Electrical and Electronics Engineers* IEEE (1990) las técnicas de software se entienden como:

> Procedimientos técnicos y gerenciales que ayudan en la evaluación y mejora de proceso de desarrollo de software. (p. 74)

Tomando en cuenta la definición anterior de lo que es una técnica en el desarrollo de software y enfocando esto a las pruebas de software puedo rescatar que, las técnicas de pruebas de software son procedimientos de diseño de pruebas que facilitan el análisis para encontrar los casos de prueba más efectivos, permitiendo llevar a cabo esta tarea de una manera más rápida y con menor esfuerzo.

4.4. Técnicas de pruebas de software que más he utilizado

Existen diferentes técnicas de pruebas de software, pero en este documento solo hago énfasis en las técnicas de pruebas de software que más he utilizado en los tres años siguientes a mi graduación de la universidad, las cuales son técnicas de pruebas funcionales de caja negra y basadas en la experiencia.

A lo largo de estos tres años trabajando en varios proyectos de software, utilicé 4 técnicas que me ayudaron mucho en el diseño de los casos de prueba. Estas técnicas me permitieron crear casos de prueba más precisos, cubriendo escenarios donde es más usual que se presenten defectos, mejorando por mucho la calidad del software donde se apliquen.

Las técnicas de pruebas de software que más utilicé fueron:

- Partición de equivalencia
- Valores límite

- Adivinar Errores
- Prueba por pares

Estas técnicas de pruebas de software me permitieron realizar pruebas de calidad, minimizando el tiempo invertido en el diseño de los casos de prueba, además me permitieron diseñar el menor número de casos de prueba, seleccionando solo los óptimos, para así identificar los defectos que mayor impacto representaron en su momento en los distintos proyectos donde las he utilizado.

4.5. Uso de técnicas de pruebas en proyectos trabajados

A continuación, presento las definiciones de las técnicas de pruebas de software mencionadas anteriormente y algunos ejemplos de cómo las he utilizado en los proyectos donde he trabajado. Es importante destacar que en cada proyecto donde he trabajado, se me ha solicitado diseñar el menor número de casos de prueba posible, cubriendo todas las posibles casuísticas. En este sentido, el uso de las técnicas de pruebas de software se convierte en una ventaja a la hora de trabajar en el diseño de los casos de prueba.

4.5.1. Partición de equivalencia (*Equivalence Partitioning*)

El *International Software Testing Qualifications Board* ISTQB (2018) menciona que una partición de equivalencia trabaja de la siguiente manera:

> La partición de equivalencia divide los datos en particiones (también conocidas como clases de equivalencia) de tal manera que se espera que todos los miembros de una partición determinada se procesen de la misma manera. (p. 57)

Esta técnica de pruebas se complementa con la técnica de pruebas Valores Límite para cubrir la mayoría de los escenarios con el menor número de casos de prueba posibles. En esta técnica de pruebas de software el *tester* asume que, cualquier elemento dentro de la partición tiene el mismo comportamiento y arroja el mismo resultado, por lo que no es necesario probar todos los elementos de la partición.

4.5.2. Experiencia en uso de la técnica Partición de Equivalencia

A continuación, presento un ejemplo de un requerimiento (**ficticio**) que me permite exponer el uso de esta técnica de pruebas de software.

"LA PRIMA TOTAL DE UNA PÓLIZA DE SEGUROS DE GASTOS MÉDICOS MAYORES CON LA COBERTURA DE ACCIDENTES PERSONALES COLECTIVOS, SE CALCULA SUMANDO LA PRIMA DE LA PÓLIZA, MÁS EL DERECHO DE PÓLIZA, MÁS EL IVA."

Por lo que la fórmula quedaría como $\boldsymbol{PT = (\Sigma PN_i + D) + iva}$

Donde:

$\boldsymbol{PT} = Prima\ Total$

$\boldsymbol{PN_i} = Prima\ Neta$ (Por cada asegurado $\boldsymbol{i}$)

$\boldsymbol{D} = Derecho\ de\ Póliza$

Donde $\boldsymbol{D}$ depende de $\boldsymbol{\Sigma PN}$ ya que puede caer en distintos escenarios

Nota: *Para dar solución a este problema se utilizó como valor %16 para el iva, impuesto que se aplica a los productos y/o servicios especificados por las leyes fiscales vigentes al momento de escribir este documento*.

Para calcular el derecho de póliza $\boldsymbol{D}$ se tiene la siguiente especificación donde $\boldsymbol{\Sigma PN}$ se menciona solo como la prima de la póliza para mejor compresión del ejemplo:

1. Si la prima es de $0 a $200, no se cobra derechos de póliza y no se manda a revisión.
2. Si la prima es mayor a $200 y menor o igual a $500, se cobran $300 por derechos de póliza y no se manda a revisión.
3. Si la prima es mayor a $500 y menor o igual a $50000, se cobran $400 por derechos de póliza y no se manda a revisión.
4. Si la prima es mayor $50000 no se debe cobrar derechos de póliza y se debe mandar a revisión.

Las particiones de equivalencia identificadas son:

- Prima menor a $0
- Prima de $0 a $200
- Prima de $201 a $500
- Prima de $501 a $50000
- Prima mayor a $50000

En este ejemplo existen 3 particiones de equivalencia válidas que requieren ser evaluadas con al menos 1 caso de prueba cada una. Esto es fundamental para asegurar que al ingresar un valor dentro de cada partición obtengamos el resultado esperado. Además, para evaluar las particiones inválidas se necesitarían 2 casos de prueba adicionales; uno con una prima menor a 0 y otro con una prima mayor a $50,000. De esta manera, se verifica el comportamiento del sistema frente a valores que no deberían ser aceptados.

El diseño de los casos de prueba utilizando la técnica **partición de equivalencias** quedaría de la siguiente manera:

DISEÑO DE CASOS DE PRUEBA CON PRIMERA TÉCNICA DE PRUEBAS					
ID de caso	Tipo de caso de prueba	Partición a la que pertenece	Valor de la prima (Valores de entrada)	Derecho de póliza	Prima Total (Resultado esperado)
TC001	Inválido	**Prima menor de $0**	-$1	NA	No debe permitir ingresar datos negativos.
TC002	Válido	**Prima de $0 a $200**	$125.00	$0.00	$145.00
TC003	Válido	**Prima mayor a $200 y menor de $500**	$250.00	$300.00	$638.00
TC004	Válido	**Prima igual o mayor a $500 y menor o igual a $50000**	$6200.00	$400.00	$7656.00
TC005	Inválido	**Prima mayor a $50000**	$50001.00	NA	Mensaje donde indica “Se mandará a revisión”

Tabla 2: Casos de prueba con técnica de pruebas Partición de Equivalencia

En la tabla anterior se observan los casos de prueba válidos e inválidos, así como también la partición de equivalencia a la que pertenecen y el resultado que se debería obtener en cada ejecución. También se observa que sólo se diseñó un caso de prueba por cada **partición de equivalencia**, para así evaluar con un solo valor el comportamiento del software sin tener que invertir demasiado esfuerzo en probar todas las combinaciones posibles en cada una de las particiones.

4.5.3. Análisis de Valor Límite (*Boundary Value Analysis*)

El *International Software Testing Qualifications Board* ISTQB (2018) menciona lo siguiente sobre esta técnica de pruebas de software:

> Análisis de valor límite (BVA por sus siglas en inglés), es una extensión de la partición de equivalencia, pero solo se puede usar cuando la partición está ordenada y consta de datos numéricos o secuenciales. Los valores mínimo y máximo (o el primer y el último valor) de una partición son sus valores límite. (p. 58)

Esta técnica de pruebas permite al *tester* identificar errores donde hay cambios de entradas al sistema. Estas entradas de datos pueden ser las particiones de equivalencia y el límite hasta donde termina o inicia una partición (estos límites pueden ser la parte donde más errores existen). Esta técnica de pruebas de software se puede utilizar cuando se tiene conocimiento de los valores de entrada de la funcionalidad, necesarios para levantar los cambios en el resultado esperado, ya sea que se realicen pruebas de caja blanca o pruebas de caja negra.

4.5.4. Experiencia en uso de la técnica Análisis de Valor Límite

A continuación, presento un ejemplo de cómo he aplicado esta técnica en el ámbito laboral.

Utilizando el mismo requerimiento mencionado anteriormente donde se calcula la prima total para una póliza se puede observar que:

1. Si la prima es de $0 a $200, no se cobra derechos de póliza y no se manda a revisión.
2. Si la prima es mayor a $200 y menor o igual a $500, se cobran $300 por derechos de póliza y no se manda a revisión.
3. Si la prima es mayor a $500 y menor o igual a $50000, se cobran $400 por derechos de póliza y no se manda a revisión.
4. Si la prima es mayor $50000 no se debe cobrar derechos de póliza y se debe mandar a revisión.

Dado lo anterior, los valores límite que se identifican son:

Prima = -1, Prima = 0, Prima = 1

Prima = 199, Prima = 200, Prima = 201

Prima = 499, Prima = 500, Prima = 501

Prima = 49999, Prima = 50000, Prima = 50001

Dada la relevancia del tipo de proyecto y para reducir los riesgos vinculados a los casos de prueba, se opta por implementar pruebas de valores límite con 3 valores que abarcan las situaciones que se encuentran por debajo del límite, en el límite y por encima del límite de cada partición de equivalencia identificada.

En el análisis anterior, se evidencia la necesidad de diseñar 12 casos de prueba (10 válidos y 2 inválidos) para abarcar el cálculo de la prima en cada límite. Esto se debe a que, según el costo de la prima de la póliza, se añade un costo adicional por el derecho de póliza, el cual puede ubicarse en diferentes particiones de equivalencia.

El diseño de los casos de prueba utilizando la técnica **análisis de valor límite** quedaría de la siguiente manera:

DISEÑO DE CASOS DE PRUEBA CON SEGUNDA TÉCNICA DE PRUEBAS					
ID de caso	Tipo de caso de prueba	Partición a la que pertenece	Valor límite (Valores de entrada)	Derecho de póliza	Prima Total (Resultado esperado)
TC006	Inválido	Prima menor de $0	**-$1**	NA	No debe permitir ingresar datos negativos.
TC007	Válido	Prima de $0	**$0**	$0	$0
TC008	Válido	Prima mayor a $0	**$1**	$0	$1.16
TC009	Válido	Prima menor a $200	**$199**	$0	$230.84
TC010	Válido	Prima de $200	**$200**	$0	$232.00
TC011	Válido	Prima mayor a $200	**$201**	$300	$581.16
TC012	Válido	Prima menor a $500	**$499**	$300	$926.84
TC013	Válido	Prima de $500	**$500**	$300	$928.00
TC014	Válido	Prima mayor a $500	**$501**	$400	$1,045.16
TC015	Válido	Prima menor a $50000	**$49999**	$400	$58,462.84
TC016	Válido	Prima de $50000	**$50000**	$400	$58,464.00
TC017	Inválido	Prima mayor a $50000	**$50001.00**	NA	Mensaje donde indica “Se mandará a revisión”

Tabla 3: Casos de prueba con técnica de pruebas Análisis de Valor Límite

En la tabla anterior, se muestran los casos de prueba que abarca las casuísticas de los valores límite identificados, donde la prima neta aumenta cuando el monto de la prima de la póliza aumenta, esto debido a que se posiciona en distintas particiones de equivalencia para el cobro del derecho de póliza. Para cada caso de prueba se busca evaluar el comportamiento con valores de entrada válidos e inválidos. Es importante entender que el enfoque de evaluar los valores límite de entrada permite identificar los errores más comunes donde se presentan cambios, ya que el desarrollador puede confundirse al programar estos cambios en el código del software. Además, dependiendo del criterio del *tester*, se determina si se evaluarán los

valores en el límite, valores arriba del límite y valores abajo del límite. Esto dependerá del riesgo de lo que se está probando y el impacto que generaría si algún límite no se evalúa.

4.5.5. Adivinar Errores (*Error Guessing*)

Para entender mejor esta técnica de pruebas de software, el *International Software Testing Qualifications Board* ISTQB (2018) nos da una pequeña explicación:

> La adivinación de errores es una técnica utilizada para anticipar la aparición de errores, defectos y fallas, basándose en el conocimiento del *tester*, incluyendo:
>
> - Cómo ha funcionado la aplicación en el pasado
> - Qué tipo de errores se suelen cometer
> - Fallos que se han producido en otras aplicaciones (p. 61)

La elección de utilizar esta técnica de pruebas se basa en la experiencia y el conocimiento que se tenga en el campo de las pruebas, así como en el entendimiento del software a ser probado. Al emplear esta técnica, se busca identificar de manera eficiente los errores más comunes o recurrentes en el software. Cabe señalar que, es recomendable utilizar esta técnica de pruebas de software después de haber aplicado técnicas de pruebas más formales como lo son Partición de equivalencia, Valores límite, Estado de transición, Tablas de decisión, etc.

Esta técnica de pruebas de software la he utilizado para identificar los diferentes escenarios que existen en base a mi experiencia como *tester* probando diferentes sistemas de software. Esta técnica requiere cierta experiencia debido a que, al realizar pruebas, existen escenarios que no se especifican en los requerimientos, pero como *tester* se deben cubrir, para dar ese plus requerido en las pruebas a realizar.

4.5.6. Experiencia en el uso de la técnica Adivinar Errores

A continuación, presento un ejemplo de cómo he aplicado esta técnica en el ámbito laboral.

Al ingresar a un nuevo sistema software, por seguridad se requiere que el usuario se identifique con Usuario y Contraseña. El sistema permite ingresar valores alfanuméricos en los campos e identifica si el usuario existe en la base de datos al momento de presionar el botón entrar. Si el usuario existe, revisa que su contraseña sea correcta, si la contraseña es correcta se permite el acceso al sistema, en caso contrario no se permite el acceso y se muestra un mensaje de error que especifique el motivo.

Para iniciar las pruebas utilizando esta técnica, es fundamental comprender el sistema que se encuentra bajo evaluación, posteriormente se procede al diseño de los casos de prueba basándose en la experiencia previa con sistemas similares. Su enfoque se realiza particularmente en las áreas donde es más frecuente encontrar errores que no se detallan en las pruebas técnicas ni en los requerimientos, de esta manera se busca abordar los posibles problemas que podrían surgir y garantizar una cobertura más completa durante las pruebas del software.

Continuando con el ejemplo, procedo a plantear las siguientes preguntas:

1. ¿Cuál sería el resultado si se ingresa solo el usuario en el campo correspondiente pero no la contraseña, y se intenta ingresar al sistema?
2. ¿Cuál sería el resultado si se ingresa solo la contraseña en el campo correspondiente pero no el usuario, y se intenta ingresar al sistema?
3. ¿Cuál sería el resultado si no se ingresa ni usuario ni contraseña en los campos correspondientes, y se intenta ingresar al sistema?
4. ¿Cuál sería el resultado si se ingresa usuario y contraseña correctos en los campos correspondientes, y se intenta ingresar al sistema?
5. ¿Cuál sería el resultado si se ingresan los valores "105 OR 1=1" para el campo usuario y se intenta ingresar al sistema?
6. ¿Cuál sería el resultado si se ingresan los valores "" or ""=" para usuario y contraseña en los campos correspondientes y se intenta ingresar al sistema?

Para el escenario de arriba puedo identificar los siguientes escenarios utilizando la técnica de pruebas adivinar errores.

1. Ingresar usuario, pero no contraseña e intentar ingresar al sistema
2. Ingresar contraseña, pero no usuario e intentar ingresar al sistema
3. No ingresar ni usuario ni contraseña e intentar ingresar al sistema
4. Ingresar usuario y contraseña válidos e ingresar al sistema
5. Ingresar valores para usuario "105 OR 1=1" e intentar ingresar al sistema
6. Ingresar " or ""=" para usuario y contraseña e intentar ingresar al sistema

Estos 6 escenarios pueden ser cubiertos por 6 casos de prueba donde los dos últimos casos de prueba son ejemplos de SQL *Injection*, y como *tester* experimentado se deben probar estos escenarios a pesar de que los requerimientos no detallen cómo debe responder el nuevo software.

El diseño de los casos de prueba utilizando la técnica **adivinar errores** quedaría de la siguiente manera:

DISEÑO DE CASOS DE PRUEBA CON TERCERA TÉCNICA DE PRUEBAS			
ID de caso	Tipo de caso de prueba	Valores de entrada	Resultado esperado
TC018	Inválido	**Usuario:** UACM **Contraseña:** vacío	Mensaje donde indica que **es necesario ingresar usuario y/o contraseña**
TC019	Inválido	**Usuario:** vacío **Contraseña:** Uacm2022	Mensaje donde indica que **es necesario ingresar usuario y/o contraseña**
TC020	Inválido	**Usuario:** vacío **Contraseña:** vacío	Mensaje donde indica que **es necesario ingresar usuario y/o contraseña**
TC021	Válido	**Usuario:** UACM **Contraseña:** Uacm2022	El sistema debe permitir el acceso al usuario
TC022	Inválido	**Usuario:** 105 OR 1=1 **Contraseña:** Uacm2022	Mensaje donde indica que **el usuario y/o contraseña son inválidos**
TC023	Inválido	**Usuario:** " or ""=" **Contraseña:** " or ""="	Mensaje donde indica que **el usuario y/o contraseña son inválidos**

Tabla 4: Casos de prueba con técnica Adivinar Errores

Como se puede observar en la tabla de arriba, la técnica de pruebas de Adivinar Errores nos permite diseñar casos de prueba en base a los requerimientos más usuales en la mayoría de los sistemas, estos casos de prueba cubren escenarios que es muy importante probar en base a la experiencia del tester y son muy rápidos de diseñar para el inicio de pruebas exploratorias.

Cabe recalcar nuevamente que esta técnica de pruebas de software requerirá el uso de técnicas más formales para complementar los resultados.

4.5.6.1. Ejemplo de diseño y ejecución de casos de prueba con técnica Adivinar Errores

A continuación, se muestra el diseño y ejecución de los casos de prueba analizados con la técnica de pruebas de Adivinar Errores de la sección anterior. Los casos de prueba se relacionan con el Id único, asignado para llevar a cabo el seguimiento y ejecución de las pruebas de un *login*.

Como no cuento con los derechos para ejecutar los casos de prueba diseñados en el sistema donde he implementado la técnica, he tomado un sistema ejemplo que he diseñado para las pruebas, el cual tiene el nombre de SBDCIM (Sistema de Base de datos de la clínica Integral de la Mujer)

Diseño y ejecución del caso de prueba TC018

Id de caso	TC018
Nombre	Intentar ingresar al sistema sin contraseña
Objetivo	Validar que el sistema no permita el acceso si no se ingresa una contraseña válida
Precondición	No aplica
Postcondición	El sistema no debe otorgar acceso
Valores de entrada	Usuario: UACM Contraseña: vacío
Pasos a seguir	1. Ejecutar la aplicación de escritorio

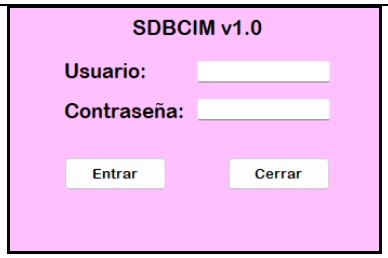
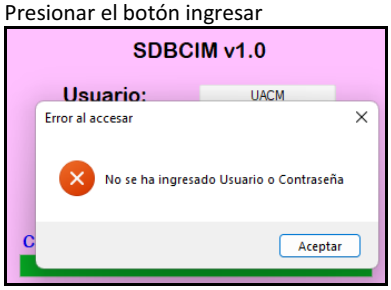
	 2. Ingresar usuario  3. Presionar el botón ingresar 
Alternativas	No aplica
Resultados esperados	El sistema muestra el mensaje “No se ha ingresado Usuario o Contraseña”
Resultados obtenidos	Se muestra el mensaje “No se ha ingresado Usuario o Contraseña”

Tabla 5: Caso de prueba 1 con técnica de Adivinar Errores

Diseño y ejecución del caso de prueba TC019

Id de caso	TC019
Nombre	Intentar ingresar al sistema sin usuario
Objetivo	Validar que el sistema no permita el acceso si no se ingresa un usuario válido
Precondición	No aplica
Postcondición	El sistema no debe otorgar acceso

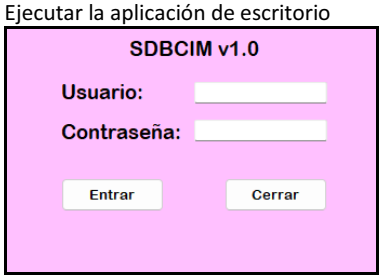
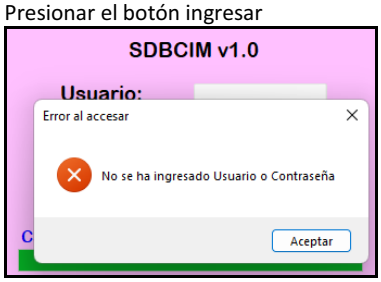
Valores de entrada	Usuario: vacío Contraseña: Uacm2022
Pasos a seguir	1. Ejecutar la aplicación de escritorio  2. Ingresar solo contraseña  3. Presionar el botón ingresar 
Alternativas	No aplica
Resultados esperados	El sistema muestra el mensaje “No se ha ingresado Usuario o Contraseña”
Resultados obtenidos	Se muestra el mensaje “No se ha ingresado Usuario o Contraseña”

Tabla 6: Caso de prueba 2 con técnica de Adivinar Errores

Diseño y ejecución del caso de prueba TC020

Id de caso	TC020
Nombre	Intentar ingresar al sistema sin usuario ni contraseña
Objetivo	Validar que el sistema no permita el acceso si no se ingresa usuario y contraseña válidos
Precondición	No aplica

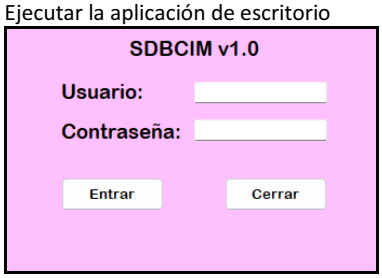
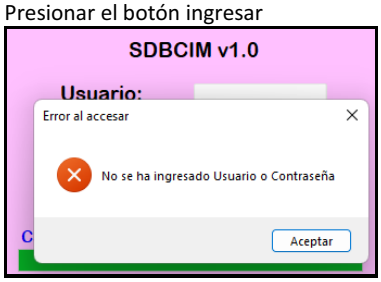
Postcondición	El sistema no debe otorgar acceso
Valores de entrada	Usuario: vacío Contraseña: vacío
Pasos a seguir	1. Ejecutar la aplicación de escritorio  2. Presionar el botón ingresar 
Alternativas	No aplica
Resultados esperados	El sistema muestra el mensaje “No se ha ingresado Usuario o Contraseña”
Resultados obtenidos	Se muestra el mensaje “No se ha ingresado Usuario o Contraseña”

Tabla 7: Caso de prueba 3 con técnica de Adivinar Errores

Diseño y ejecución del caso de prueba TC021

Id de caso	TC021
Nombre	Intentar ingresar al sistema con usuario y contraseña válidos
Objetivo	Validar que el sistema permita el acceso si se ingresa usuario y contraseña válidos
Precondición	No aplica
Postcondición	El sistema debe otorgar acceso y mostrar la pantalla principal con las diferentes opciones
Valores de entrada	Usuario: UACM Contraseña: Uacm2022
Pasos a seguir	1. Ejecutar la aplicación de escritorio

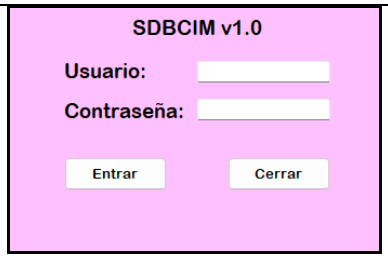
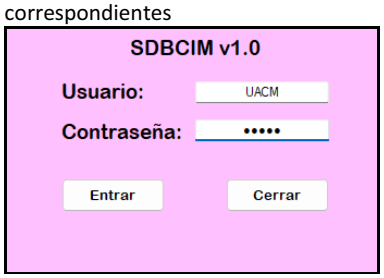
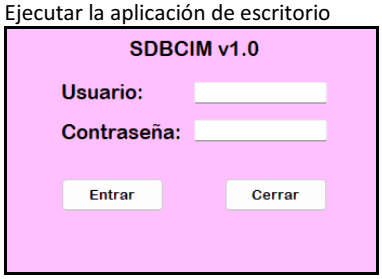
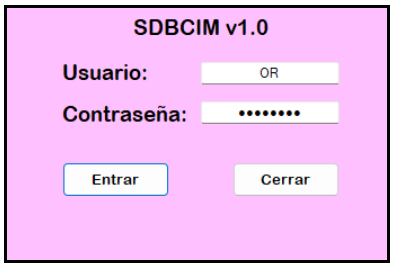
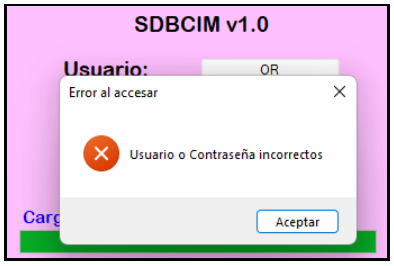
	 2. Ingresar Usuario y Contraseña en los campos correspondientes  3. Presionar el botón ingresar 
Alternativas	No aplica
Resultados esperados	El sistema muestra la pantalla principal con las diferentes opciones disponibles.
Resultados obtenidos	El sistema permite el acceso al usuario y muestra la pantalla principal de SBDCIM con las diferentes opciones.

Tabla 8: Caso de prueba 4 con técnica de Adivinar Errores

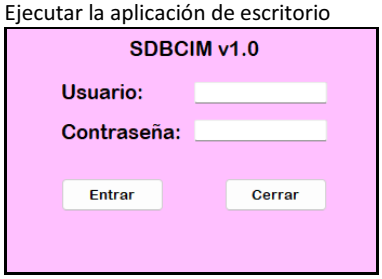
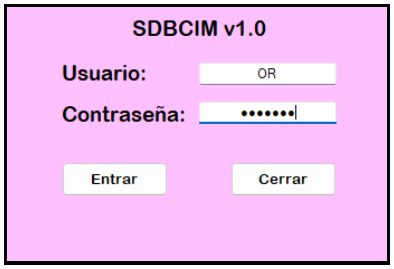
Diseño y ejecución del caso de prueba TC022

Id de caso	TC022
Nombre	Intentar ingresar al sistema con usuario inválido y contraseña válida
Objetivo	Validar que el sistema no permita SQL *Injection*
Precondición	No aplica
Postcondición	El sistema no debe otorgar el acceso
Valores de entrada	Usuario: 105 OR 1=1 Contraseña: Uacm2022
Pasos a seguir	1. Ejecutar la aplicación de escritorio  2. Ingresar Usuario y Contraseña en los campos correspondientes  **Nota:** *El sistema SBDCIM cuenta con restricciones por lo que solo permite ingresar letras en el campo "Usuario"* 3. Presionar el botón ingresar 

Alternativas	No aplica
Resultados esperados	El sistema muestra el mensaje “Usuario o Contraseña incorrectos” y no permite el acceso
Resultados obtenidos	Se muestra el mensaje “Usuario o Contraseña incorrectos” negando el acceso

Tabla 9: Caso de prueba 5 con técnica de Adivinar Errores

Diseño y ejecución del caso de prueba TC023

Id de caso	TC023
Nombre	Intentar ingresar al sistema con usuario y contraseña inválidos
Objetivo	Validar que el sistema no permita SQL *Injection*
Precondición	No aplica
Postcondición	El sistema no debe otorgar el acceso
Valores de entrada	Usuario: "or ""=" Contraseña: "or ""="
Pasos a seguir	1. Ejecutar la aplicación de escritorio  2. Ingresar Usuario y Contraseña en los campos correspondientes  **Nota:** *El sistema SBDCIM cuenta con restricciones por lo que solo permite ingresar letras en el campo “Usuario”* 3. Presionar el botón ingresar

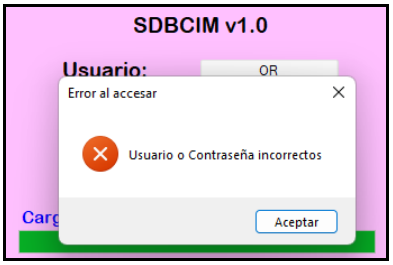
	
Alternativas	No aplica
Resultados esperados	El sistema muestra el mensaje “Usuario o Contraseña incorrectos” y no permite el acceso
Resultados obtenidos	Se muestra el mensaje “Usuario o Contraseña incorrectos” negando el acceso

Tabla 10: Caso de prueba 6 con técnica de Adivinar Errores

Durante el diseño y la ejecución de los casos de prueba, es común encontrar situaciones en las que no se pueden ingresar estrictamente todos los datos de entrada como se indica en el análisis y diseño inicial. Esto se debe a las restricciones presentes en muchos sistemas, que limitan la entrada de ciertos datos en determinados campos. En estos casos, el *tester* debe evaluar si el caso de prueba se considera exitoso o fallido basándose en los resultados obtenidos y las restricciones que se aplican. Además, debe analizar si estas restricciones permiten concluir que los casos de prueba están cubiertos, ya que las restricciones de valores de entrada aceptables protegen al sistema contra posibles fallas incontroladas.

Es crucial documentar y mostrar todas las pruebas realizadas como evidencia, demostrando el comportamiento del sistema al ingresar los valores especificados en cada caso de prueba.

4.5.7. Tablas de Decisión (*Decision Table*)

La forma de trabajar de esta técnica de pruebas de software según el *International Software Testing Qualifications Board* ISTQB (2018), se comprende de la siguiente manera:

Tablas de decisiones son una buena manera de registrar reglas comerciales complejas que un sistema debe implementar. Al crear tablas de decisiones, el *tester* identifica las condiciones (entradas) y las acciones resultantes (salidas) del sistema. Estos forman las filas

de la tabla, generalmente con las condiciones en la parte superior y las acciones en la parte inferior. Cada columna corresponde a una regla de decisión que define una combinación única de condiciones que da como resultado la ejecución de las acciones asociadas con esa regla. Los valores de las condiciones y acciones generalmente se muestran como valores booleanos (verdadero o falso) o valores discretos (por ejemplo, rojo, verde, azul), pero también pueden ser números o rangos de números. Estos diferentes tipos de condiciones y acciones pueden encontrarse juntos en la misma tabla. (p. 58)

Esta técnica de pruebas de software es una excelente técnica de pruebas donde podemos especificar todas las reglas de negocio que deseemos tomar en cuenta, las condiciones que se deben cumplir y las acciones que se llevan a cabo cuando se cumplen estas.

4.5.8. Experiencia con el uso de la técnica Tablas de decisión

En el siguiente ejemplo, compartiré mi experiencia en el uso de esta técnica de pruebas de software en un proyecto específico.

El objetivo es evaluar las condiciones necesarias para permitir que un usuario acceda a un sistema.

UN SISTEMA BANCARIO SOLO PERMITE EL ACCESO A PERSONAL REGISTRADO, POR LO QUE EL USUARIO DEBE INGRESAR NOMBRE, CORREO DEL BANCO Y SU TOKEN PERSONAL. SI EL NOMBRE DE USUARIO O CORREO NO EXISTEN EN LA BASE DE DATOS O SI INGRESA MAL LA INFORMACIÓN, EL SISTEMA NO PERMITE EL ACCESO Y POR SEGURIDAD NO INDICA INFORMACIÓN DEL ERROR.

Para dar solución al requerimiento, se desarrolla la siguiente tabla de decisión con la información indicada.

	Regla 1	Regla 2	Regla 3	Regla 4	Regla 5	Regla 6	Regla 7	Regla 8
Input								
Nombre	F	V	F	V	F	V	F	V
Email	F	F	V	V	F	F	V	V
Token	F	F	F	F	V	V	V	V
Output								
Acceso	F	F	F	F	F	F	F	V

Tabla 11: Tabla de decisión de casos de prueba

Por lo que los casos de prueba quedarían de la siguiente manera:

1. Si el nombre, el email y el token no son correctos, **no se permite el acceso**.
2. Si el nombre es correcto, pero el email y el token no son correctos, **no se permite el acceso**.
3. Si el nombre no es correcto, el email es correcto, pero el token no es correcto, **no se permite el acceso**.
4. Si el nombre y el email son correctos pero el token no es correcto, **no se permite el acceso**.
5. Si el nombre y el email no son correctos, pero el token es correcto, **no se permite el acceso**.
6. Si el nombre es correcto, el email no es correcto, pero el token es correcto, **no se permite el acceso**.
7. Si el nombre no es correcto, el email y el token son correctos, **no se permite el acceso**.
8. Si el nombre es correcto, el email es correcto y el token es correcto, **se permite el acceso**.

Cabe señalar que los casos de prueba que se diseñan con esta técnica de pruebas de software se apegan a los requerimientos de usuario, cumpliendo las reglas de negocio que se presentan para su análisis. Es importante entender que las salidas siempre serán en base los factores de entrada y las reglas, si hay cambios en las reglas de negocio o en las entradas, varios casos de

prueba se pueden ver impactados por lo que se recomienda usar esta técnica de pruebas de software cuando se tenga certeza de que los cambios antes mencionados no serán constantes.

4.5.9. Técnica de pruebas Prueba por Pares (*Pairwise Testing*)

El uso de esta técnica de pruebas de software según el *International Software Testing Qualifications Board* ISTQB (2019) se explica a continuación:

> Prueba por pares se utiliza cuando se prueba software en el que varios parámetros de entrada, cada uno con varios valores posibles, deben probarse en combinación, lo que da lugar a más combinaciones de las que es factible probar en el tiempo permitido. (p. 31)

Esta técnica se basa en probar pares de valores de entrada disminuyendo el número de casos de prueba necesarios para cubrir la mayor cantidad de escenarios posibles. Además, se requiere conocer los valores de entrada necesarios y aceptables. Si la cantidad de datos de entrada es baja, se necesitarán menos casos de prueba. Sin embargo, si existen numerosas combinaciones posibles de valores de entrada al sistema, el número de casos de prueba aumentará considerablemente.

Esta técnica de pruebas de software la he utilizado para determinar el menor número de casos de prueba posibles, necesarios para probar escenarios con diferentes combinaciones de entrada donde es más probable encontrar errores.

4.5.10. Experiencia en uso de técnica Prueba por Pares

A continuación, presento un ejemplo de cómo he aplicado esta técnica en el ámbito laboral.

Se requiere probar el cálculo de primas del sistema de Seguros de Gastos Médicos Mayores como en el ejemplo de la sección 4.5.2. pero para accidentes personales individuales. El sistema solicita la siguiente información del asegurado como datos de entrada para calcular la prima individual del seguro.

1. Género
2. Ciudad
3. Ocupación
4. Tipo de riesgo por ocupación
5. Tipos de hospital en convenio

En base a esas variables de entrada se pueden seleccionar los siguientes valores interesantes:

- Género: {Masculino, Femenino}
- Ciudad: {CDMX, Monterrey, Guadalajara}
- Ocupación: {Estudiante, Empleado, Obrero, Empresario}
- Riesgo: {A, B, C}
- Hospital: {Básico, Premium, Platino}

Si se desarrollaran todos los pares que se pueden obtener de las diferentes combinaciones serían demasiados casos de prueba. El producto cartesiano de todos los valores interesantes de entradas sería en total **2 × 3 × 4 × 3 × 3 = 216 casos de prueba**.

Para obtener una buena combinación por pares que agregue más valor a las pruebas, podemos utilizar diferentes softwares que permiten ahorrar tiempo en el diseño de las combinaciones para los casos de prueba.

En este ejemplo utilizaré la herramienta online llamada **pairwiseTool** de Victor Dementiev. Esta herramienta se encuentra disponible en el siguiente enlace o *link*: https://pairwise.teremokgames.com/

Al ingresar en la dirección web de la herramienta la pantalla de inicio que muestra es la siguiente (ver Imagen 2).

Imagen 2: PairwiseTool en línea

Para empezar a trabajar en la herramienta solo es necesario especificar las variables de entrada en cada columna de la tabla (ver Imagen 3).

Imagen 3: Variables de entrada en PairwiseTool

Ahora se llenan los valores de las variables de entrada en la herramienta para después proceder a realizar las combinaciones por pares con un solo clic como se ve en la Imagen 4.

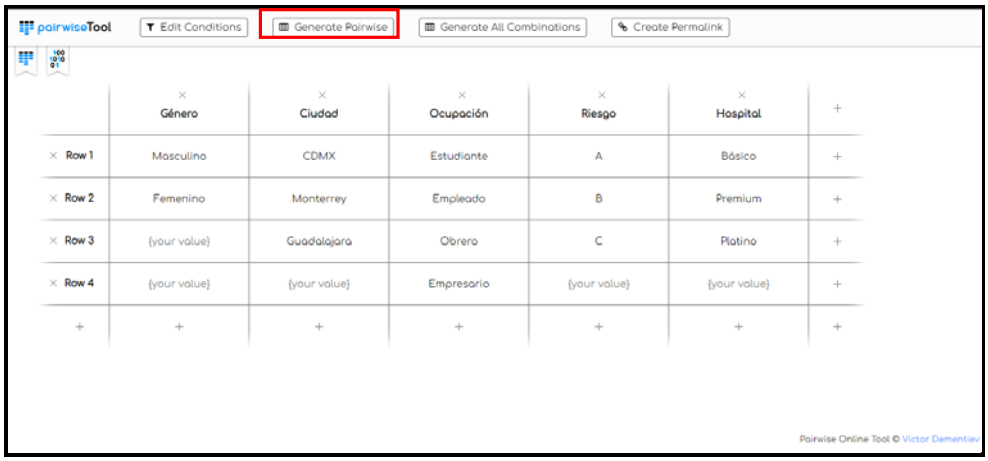


Imagen 4: Llenado de variables para casos de prueba combinación por pares

La herramienta obtiene las combinaciones por pares más importantes y genera un archivo Excel con el resultado como se observa en la imagen 5.

	A	B	C	D	E	F	G	H	I	J	K	L
1		**Género**	**Ciudad**	**Ocupación**	**Riesgo**	**Hospital**						
2	**1**	Masculino	Monterrey	Empleado	B	Premium						
3	**2**	Masculino	Guadalajara	Obrero	C	Platino						
4	**3**	Femenino	Monterrey	Obrero	A	Básico						
5	**4**	Femenino	Guadalajara	Empresario	A	Premium						
6	**5**	Femenino	CDMX	Estudiante	B	Platino						
7	**6**	Femenino	CDMX	Empleado	C	Básico						
8	**7**	Masculino	Guadalajara	Estudiante	C	Básico						
9	**8**	Masculino	CDMX	Empleado	A	Premium						
10	**9**	Masculino	CDMX	Obrero	A	Platino						
11	**10**	Masculino	Monterrey	Empresario	B	Básico						
12	**11**	Femenino	CDMX	Obrero	B	Básico						
13	**12**	Femenino	CDMX	Empresario	C	Premium						
14	**13**	Femenino	Monterrey	Estudiante	A	Platino						
15	**14**	Femenino	Guadalajara	Empleado	A	Básico						

Imagen 5: Combinaciones para casos de prueba por pares

Es evidente que se produce un ahorro significativo en los casos de prueba, ya que, según la teoría de errores, aquellos que tienen una baja probabilidad de encontrar fallos aportan poco valor a las pruebas. En este caso, se logró ahorrar un total de 202 casos de prueba (**216 - 14**). Al enfocarnos únicamente en los 14 casos de prueba necesarios, se cubren la mayoría de los escenarios importantes de manera efectiva.

Si se tiene tiempo suficiente para invertir en las pruebas, se puede optar por la opción de realizar los casos de prueba de todas las combinaciones posibles (*All pairwise*). La herramienta nos permite generar todas estas combinaciones con un solo clic en la opción de la parte superior de la herramienta (ver Imagen 6).

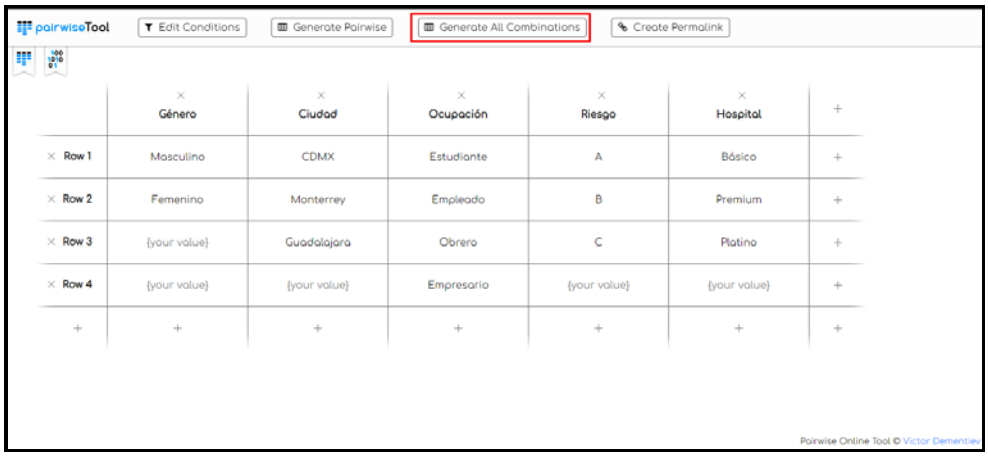
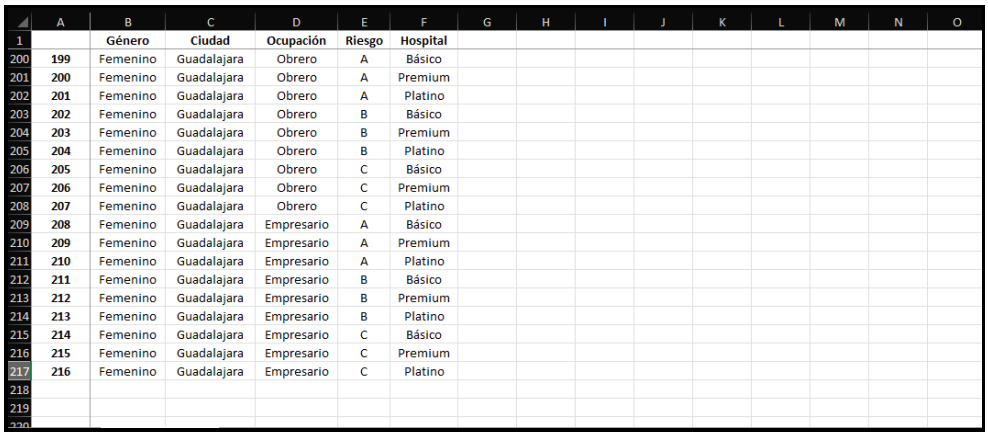


Imagen 6: Opción para generar todas las combinaciones posibles

El resultado fue el siguiente:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Género	Ciudad	Ocupación	Riesgo	Hospital									
200	199	Femenino	Guadalajara	Obrero	A	Básico									
201	200	Femenino	Guadalajara	Obrero	A	Premium									
202	201	Femenino	Guadalajara	Obrero	A	Platino									
203	202	Femenino	Guadalajara	Obrero	B	Básico									
204	203	Femenino	Guadalajara	Obrero	B	Premium									
205	204	Femenino	Guadalajara	Obrero	B	Platino									
206	205	Femenino	Guadalajara	Obrero	C	Básico									
207	206	Femenino	Guadalajara	Obrero	C	Premium									
208	207	Femenino	Guadalajara	Obrero	C	Platino									
209	208	Femenino	Guadalajara	Empresario	A	Básico									
210	209	Femenino	Guadalajara	Empresario	A	Premium									
211	210	Femenino	Guadalajara	Empresario	A	Platino									
212	211	Femenino	Guadalajara	Empresario	B	Básico									
213	212	Femenino	Guadalajara	Empresario	B	Premium									
214	213	Femenino	Guadalajara	Empresario	B	Platino									
215	214	Femenino	Guadalajara	Empresario	C	Básico									
216	215	Femenino	Guadalajara	Empresario	C	Premium									
217	216	Femenino	Guadalajara	Empresario	C	Platino									
218															
219															

Imagen 7: Casos de prueba con todas las combinaciones posibles

El archivo Excel generado contiene todas las combinaciones por pares que se pueden obtener con los valores de entrada para cada variable en la herramienta, en total generó **216** combinaciones posibles. Realizar estas combinaciones manualmente sería un trabajo agotador, por lo tanto, esta herramienta es una muy buena opción para trabajar con esta técnica de pruebas.

Actualmente, en el mercado existen diversas herramientas, (como la mencionada en este ejemplo) que facilitan la identificación de las posibles combinaciones. Estas herramientas ofrecen opciones avanzadas, como la capacidad de excluir combinaciones inválidas en nuestras pruebas. La elección de la herramienta adecuada para el diseño de los casos de prueba dependerá de la magnitud del proyecto y de las características específicas requeridas.

Capítulo 5: Pruebas manuales y pruebas automatizadas

5.1. La importancia de aplicar pruebas manuales

Dentro del mundo de las pruebas de software, existen dos maneras de llevar a cabo la ejecución de estas, la primera son las pruebas manuales y la segunda son las pruebas automatizadas que ejecuta una computadora.

Las pruebas manuales desempeñan un papel fundamental durante la fase de ejecución, ya que se asemejan al uso normal que un usuario daría al nuevo software. El *tester* asume el rol de un usuario real y lleva a cabo las pruebas paso a paso, analizando y registrando los resultados de cada acción. Estos registros se documentan en evidencias de ejecución, las cuales se adjuntan posteriormente a un informe que contiene información relevante. De esta manera, se demuestra el progreso y los resultados de las pruebas, brindando una visión clara a los interesados del proyecto.

Las pruebas manuales por lo general se realizan en todos los proyectos de desarrollo de software y se aplican a todos los componentes, estas deben realizarse al menos una vez dentro de un ciclo de pruebas ya sea por un *tester* o un desarrollador. Además, estas se realizan para garantizar que los componentes del software cumplen con los requerimientos especificados por los interesados del proyecto.

En este tipo de pruebas siempre está presente el criterio del *tester*, para determinar defectos que solo pueden ser identificados si se simula ser el usuario final.

Es muy importante entender que las pruebas manuales pueden ser muy costosas ya que el esfuerzo es proporcional al tamaño del software a probar. Esto también depende del alcance y de la decisión de probar todos los flujos disponibles.

Siempre será necesario ejecutar por lo menos una vez un ciclo de pruebas manuales que confirmen que el software funciona correctamente, garantizando que el usuario final puede utilizarlo sin problema.

5.2. La importancia de aplicar pruebas automatizadas

Las pruebas automatizadas juegan un papel muy importante a la hora de probar software, estas permiten realizar las pruebas más precisas y rápidas. Además, las tareas repetitivas se ejecutan

tantas veces como sea necesario, sin requerir el esfuerzo humano, lo cual permite un ahorro de tiempo, dinero y esfuerzo. Como dice Toledo (2014) no creemos que la automatización sea capaz de sustituir el trabajo de los *testers*, sino de lograr que sea mucho mejor y de mayor alcance… (p. 129)

Las pruebas automatizadas tienen como objetivo agilizar y mejorar el proceso de pruebas, reduciendo los errores que pueden ocurrir durante su ejecución. Este tipo de pruebas son especialmente útiles para tareas repetitivas que requieren un esfuerzo considerable y suelen volverse tediosas y aburridas para los testers, lo que puede llevar a una disminución del interés y la atención necesaria para ser escéptico y detectar posibles fallos.

Aunque la automatización de pruebas no garantiza un software libre de errores, sí cubre numerosos escenarios que pueden verse afectados por los cambios introducidos al incorporar nuevas funcionalidades.

Algunos puntos importantes de los beneficios de la automatización de las pruebas son los siguientes:

1. **Regresión**. Automatizar numerosos escenarios de prueba, (o por lo menos los más relevantes) con el fin de realizar regresiones periódicas durante cada integración con dichos cambios, asegura que el software sigue funcionando correctamente y cumple con la mayoría de las funcionalidades solicitadas en los requerimientos.
2. **Agilidad**. Las pruebas automatizadas permiten acelerar la entrega continua de nuevas funcionalidades. Esto se debe a que, en las nuevas integraciones de software, se requiere la ejecución de pruebas rápidas que garanticen el correcto funcionamiento del software actualizado.
3. **Insumos**. Tener una automatización que genere los insumos requeridos en las pruebas, agiliza la manera en que se realizan las pruebas y acelera el proyecto en general. Esto se debe a que, en muchos proyectos de software se requiere gran cantidad de insumos, necesarios para realizar varios tipos de pruebas. En muchos casos es un proceso muy largo el que se debe seguir para generar estos insumos, lo que disminuye la velocidad con la que se pretende entregar software, por lo tanto, el *tester* y otros integrantes del proyecto se ven afectados.

5.2.1. Ejemplo viable de automatización de pruebas

Un ejemplo muy común es la automatización de pruebas funcionales web. En este ejemplo la automatización es viable si:

- **Ha pasado correctamente las pruebas manuales por lo menos una vez.**
- **Cumple con la mayoría de los requerimientos solicitados.**
- **Tiene una tasa de defectos muy baja.**
- **No tendrá cambios drásticos en sus elementos de la interfaz gráfica de usuario.**
- **No tiene tantas restricciones y permisos que bloqueen o no permitan usar software para pruebas automatizadas.**
- **Es estable y accesible para la ejecución de las pruebas automatizadas.**
- **Tiene disponibilidad para atender varias peticiones a la vez.**
- **Cuenta con la aceptación del usuario final.**

Si cumple con la mayoría de estos criterios, se puede decir que es viable la automatización del software.

También es importante destacar que una automatización de pruebas funcionales web, diseñada para detectar diversos tipos de errores externos al software y abordar la mayoría de ellos, puede considerarse como una solución robusta que supera muchos de los desafíos comunes en las pruebas automatizadas de este tipo. Si, además, el software de automatización de pruebas es capaz de determinar cuándo repetir las pruebas para verificar una ejecución correcta sin posibles afectaciones, deteniendo el ciclo hasta alcanzar un resultado satisfactorio o un fallo aceptable con una razón clara, entonces puede considerarse aún más robusto. No obstante, lograr esto requerirá una inversión de tiempo considerable en el diseño e implementación de la lógica necesaria para identificar errores externos, ajenos al software en pruebas.

5.3 ¿Las pruebas automatizadas sustituyen a las pruebas manuales?

Esta pregunta es muy amplia y podría tener diferentes respuestas desde el punto de vista de cada *tester.* A continuación, expongo mi punto de vista sobre este tema muy delicado.

Las pruebas manuales deben ejecutarse primero para determinar si el software a automatizar es candidato y tiene cierta madurez, permitiendo que sus regresiones puedan ser ejecutadas por un ordenador sin tanta complicación. Las pruebas automatizadas son pasos consecutivos que son ejecutados por la computadora y ahí no hay ningún razonamiento como el que realiza el *tester* manual al momento de ejecutar las pruebas. Si el *script* de la automatización no se ejecuta por algún motivo o esta falla sin ni siquiera ejecutarse, la computadora determina que el software tiene errores y se reporta como caso de prueba fallado, sin ir al fondo del problema para intentar dar solución antes de ser reportado. Por tal motivo, es necesario ejecutar pruebas manuales primero y si el software es apto para ser automatizado, se procede a automatizar los diferentes escenarios probados previamente.

En base a mí experiencia, las pruebas de software manuales no pueden ser sustituidas con las pruebas automatizadas, al contrario, estas se complementan y en conjunto cumplen el principal objetivo que es entregar software de calidad.

Capítulo 6: Conclusión

6.1. Conclusión sobre las técnicas de pruebas de software utilizadas

Las pruebas en un proyecto de desarrollo de software juegan un papel fundamental para garantizar la calidad, ya que disminuyen el número de defectos presentes, permitiendo que el nuevo software alcance cierta madurez, además de satisfacer las necesidades de los interesados del proyecto. El *tester* debe estar involucrado en etapas tempranas del desarrollo de software para así identificar los defectos desde la raíz y disminuir el impacto que conllevan, o evitar defectos consecuencia de otros.

Siempre es crucial realizar pruebas de calidad utilizando las diversas herramientas disponibles en la actualidad, lo cual permite a los *testers* desempeñar su trabajo de manera eficiente y rápida. Es importante destacar que existen distintas técnicas de pruebas de software que facilitan el diseño de casos de prueba y permiten al *tester* elegir la mejor opción para cada proyecto en particular. Asimismo, el *tester* es responsable de llevar a cabo pruebas de calidad que permitan identificar la mayor cantidad de defectos en cada ciclo de pruebas. Por lo tanto, es fundamental que el *tester* esté familiarizado con los diferentes tipos de técnicas de pruebas disponibles en la actualidad, así como con su aplicación y selección adecuada en función de las funcionalidades que se desean probar.

Las técnicas de pruebas de software expuestas en este documento son un claro ejemplo de cómo estas ayudan al *tester* a diseñar los casos de prueba apropiados, enfocándose en probar las partes del software donde es más común encontrar defectos, (Partición de equivalencia, Valores Límite, Adivinar Errores) además de ayudarlo a determinar el menor número de casos de prueba (Prueba por Pares) que cubren la mayoría de los escenarios posibles.

6.2. Conclusión sobre cuándo automatizar y cuándo no automatizar las pruebas de software

Es fundamental realizar pruebas manuales al menos una vez durante todo el ciclo de vida de pruebas. Esto permite que el *tester* identifique los defectos más comunes al simular el uso normal que un usuario daría al sistema. A su vez, las pruebas manuales allanan el camino para las pruebas automatizadas, las cuales validan la madurez del sistema para su automatización, o al menos, aseguran que las pruebas automatizadas sean correctas. Además, las pruebas manuales son importantes para validar la estabilidad del nuevo software, lo que ayuda a minimizar los problemas que podrían afectar una automatización eficiente.

El *tester* automatizador, debe conocer e identificar las diferentes herramientas que existen para automatizar pruebas y así proponer la mejor opción en el proyecto de software en el que participa. Además, siempre debe revisar y analizar la viabilidad de realizar una automatización, debido a que, es posible que se pierda más de lo que se podría ahorrar, ya que existen softwares que por su naturaleza no permiten la automatización o no cuentan con cierta madurez para ser automatizados. Un ejemplo de ello es que, si la interfaz gráfica de un sistema web tendrá demasiados cambios, sería mucho esfuerzo estar modificando todos los elementos en el *script* de automatización para poder absorber estos cambios, lo cual no es factible.

En los 3 años laborados en un proyecto, tuve la oportunidad de diseñar e implementar pruebas automatizadas, utilizando algunas de las herramientas más populares en el mercado y el patrón de diseño POM por sus siglas en inglés *Page Object Model*. Estas pruebas automatizadas eran enfocadas en las pruebas de regresión para sistemas web, donde se ejecutaban sets de pruebas que contenían varios escenarios con casos de pruebas bien diseñados, simulando el uso que un usuario le daría al nuevo software. Al realizar la automatización de pruebas tuve que superar varios retos a los que se debe enfrentar un *tester* automatizador, y con la experiencia que he obtenido puedo identificar cuando es viable realizar una automatización y cuando no, por lo

que concluyo que esto siempre quedará a criterio del tester y de la experiencia que ha tenido implementado pruebas automatizadas.

Bibliografía

International Software Testing Qualifications Board ISTQB. (2018). *Certified Tester Foundation Level Syllabus*, Version 3.1. istqb.org. Recuperado el día 11 de febrero del 2022 de https://www.istqb.org/certifications/certified-tester-foundation-level-v3-1

International Software Testing Qualifications Board ISTQB. (2019). *Certified Tester Advanced Level Syllabus*, Version 3.1.1. istqb.org. Recuperado el día 11 de febrero del 2022 de https://www.istqb.org/certifications/test-analyst

Institute of Electrical and Electronics Engineers IEEE. (1990). *Standard Glossary of Software Engineering Terminology.* IEEE Std 610.12-1990: New York. *Institute of Electrical and Electronics Engineers*.

Hamilton, T. (2023). *What is software testing? Definition*. Recuperado el día 17 de septiembre de 2023 de https://www.guru99.com/software-testing-introduction-importance.html

Olvera, E. (s.f.). Reporte del Mercado Laboral de TI en LATAM 2023. Recuperado el día 17 de septiembre de 2023 de https://hireline.io/remoto/estudio-mercado-laboral-y-empleos-de-ti-latam?year=2023

Toledo, F. (2014). Introducción a las Pruebas de Sistemas de la Información. Montevideo. Abstracta