AUTOMATIZACIÓN Y REDUCCIÓN DE TIEMPOS EN PROCESO DE PRUEBAS FORMALES PARA CERTIFICACIÓN DE SOFTWARE EMBEBIDO EN LA INDUSTRIA DE LA AVIACIÓN

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS INTELIGENTES MULTIMEDIA

PRESENTA

I.M. EVER EMMANUEL ZÚÑIGA ALCANTAR

QUERÉTARO, QUERÉTARO, SEPTIEMBRE 2017

AGRADECIMIENTOS

A mis padres por ser el pilar fundamental en todo lo que soy, en toda mi educación, tanto académica, como de la vida, por su incondicional apoyo perfectamente mantenido a través del tiempo.

A mis hermanos y sobrinas por estar siempre apoyándome y haciendo más llevadero el proceso de desarrollo de este trabajo, así como de toda la maestría.

A mi pareja, que me apoyo desde el principio del desarrollo de este trabajo, alentándome a terminarlo y estando a mi lado mostrándome su apoyo incondicional.

A mis amigos que estuvieron conmigo durante toda la maestría, ya sea en el aula o fuera de ella, por las risas y anécdotas que compartimos juntos.

Por último, pero no menos importante, a mis maestros, asesores y revisores, por apoyarme durante todo el curso de la maestría hasta su culminación con este trabajo.

Todo este trabajo ha sido posible gracias todos los anteriormente mencionados.

RESUMEN

En este documento se describe el proceso de implementación y puesta en marcha de una mejora y optimización de tiempos en el proceso de pruebas formales para la certificación de software en la industria de la aviación.

El estado del arte de este trabajo comprende normas y lineamientos de las agencias reguladoras, tipos de pruebas de software, el proceso de pruebas formales, así como metodologías de desarrollo de software.

Para el desarrollo de este trabajo es necesario indagar a fondo el proceso de pruebas formales e identificar los cuellos de botella dentro del proceso, así como el proceso de pensamiento por el cual se llegó a la idea final de mejora del proceso de pruebas formales para la certificación de software en la industria de la aviación, ambos temas se explican en el capítulo de procedimiento de investigación.

La conclusión de este trabajo es la implementación de una herramienta que ayuda a reducir cuellos de botella y tiempo de ejecución en el proceso de pruebas formales, dicha herramienta logra reducir en gran medida los tiempos y recursos usados para dicho proceso.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

GLOSARIO

Acrónimo	Definición
FADEC	Full Authority Digital Electronic Control
EEC	Electronic Engine Controller
JAVA	Lenguaje de programación enfocado a objetos
MATLAB	Matrix Laboratory
SCADE	Programa para desarrollo de software en base a bloques
ANSYS	Programa para el diseño, análisis y simulación de diferentes tipos de software
DO-178B	Estándar para el desarrollo de software en el sector de seguridad crítica de la aviación
RCTA	Radio Technical Commission for Aeronautics
SRD	Software requirements data
SDD	Software design description
SVCP	Software verification cases and procedures
SVR	Software verification results
SCI	Software configuration index
SECI	Software life cycle environment configuration index
SQAR	Software quality assurance records
SCR	Software conformity review
SAS	Software accomplishment summary
Junit	Entorno de pruebas para Java
PHPUnit	Sistema para la realización pruebas unitarias en PHP
CPPUnit	Versión del framework para lenguajes C/C++
Nunit	Versión del framework para la plataforma
GE	General Electric
Csv	Comma separated values
ACARS	Aircraft Communications Addressing and Reporting System
ARINC	Aeronautical Radio Inc.
VHF	Very High Frequency
ATIS	Servicio automático de información terminal
SIGMET	Información meteorológica significativa
CDU	Control de unidades de visualización
PC	Personal Computer
FAA	Federal Aviation Authority
EASA	European Aviation Safety Agency

AUTOMATIZACIÓN Y REDUCCIÓN DE TIEMPOS EN PROCESO DE PRUEBAS FORMALES PARA CERTIFICACIÓN DE SOFTWARE EMBEBIDO EN LA INDUSTRIA DE LA AVIACIÓN

1. INTRODUCCIÓN

1.1. ANTECEDENTES

El proceso de pruebas formales para la certificación de software en la industria de la aviación conlleva un costo alto en cuanto al tiempo de ejecución, tardando en promedio tres semanas de desarrollo para una sola persona, esto teniendo en cuenta un promedio de número de pruebas a correr igual a cincuenta.

Todas las pruebas deben ser ejecutadas para lograr la certificación, así como evidencia de que las pruebas se corrieron, lo cual se comprueba mediante documentos que deben ser generados para evidenciar que las pruebas cubren en su totalidad los posibles escenarios de los distintos tipos de pruebas de un diseño.

1.2. DEFINICIÓN DEL PROBLEMA

Automatizar y reducir tiempos en el proceso de pruebas formales en la industria de la aviación para reducir el coste del producto, así como el error humano.

1.3. JUSTIFICACIÓN

Dicha automatización y reducción de tiempos permitirá que el proceso sea más corto y que la certificación se alcance con menos problemas, esto conlleva a que el tiempo ahorrado se puede invertir en otros programas u otras implementaciones.

1.4. OBJETIVOS

1.4.1. Objetivo general

Automatizar y reducir los tiempos de ejecución del proceso de pruebas formales del software de aviación.

1.4.2. Objetivos específicos

- Analizar el proceso de pruebas formales para encontrar cuellos de botella y reducirlos.
- Crear una herramienta que automatice parte del proceso de pruebas formales del software de aviación.

1.5. HIPÓTESIS

- El tiempo que conlleva el proceso de pruebas formales del software de aviación puede ser reducido en un 50% por medio de la generación de una herramienta que automatice parte del proceso.

- El tiempo que conlleva el proceso de pruebas formales del software de aviación puede ser reducido si se modifica la herramienta que genera los archivos de salidas.

- El tiempo que conlleva el proceso de pruebas formales del software de aviación puede ser reducido si se genera una herramienta que revise y regenere los archivos de salidas en caso de contener errores.

1.6. ORGANIZACIÓN DEL DOCUMENTO

Este documento se compone de 7 capítulos, el contenido de cada capítulo se describe a continuación con el fin de que el lector tenga una idea general de donde buscar la información que requiera (Omitiendo el actual capítulo de Introducción y el capítulo de referencias bibliográficas).

1.6.1. Capítulo 2: Marco Teórico

El capítulo de marco teórico contiene información referente al estado del arte de este trabajo, dentro de los campos más importantes se incluyen tipos de pruebas de software de aviación, conceptos básicos del desarrollo de software de aviación, regulaciones y certificaciones, etapas del desarrollo de software, etc.

1.6.2. Capítulo 3: Procedimiento de investigación

El capítulo de procedimiento de investigación abarca todo el proceso de desarrollo de una mejora al proceso de pruebas formales en la industria de la aviación.

Dicho capitulo contiene temas como, por ejemplo:

- Pasos a seguir dentro del proceso de pruebas formales
- Que son las pruebas formales
- Cómo surgió la idea de mejora
- Conceptos de mejora propuesta
- Pros y contras de cada concepto
- Desarrollo de idea a implementar y porque se eligió

1.6.3. Capítulo 4: Resultados

El capítulo de resultados contiene los efectos de la implementación de la mejora propuesta en el capítulo 3 con temas como porcentaje de mejora y su impacto dentro de la compañía.

1.6.4. Capítulo 5: Conclusiones

Este capítulo incluye las conclusiones finales a las que llevo el desarrollo de este trabajo, así como la situación actual de GE y como o el concepto elegido entre todos los mostrados, ayudo a mejorar procesos en GE al ser aplicado.

2. MARCO TEÓRICO

2.1. ¿QUÉ ES SOFTWARE EMBEBIDO?

El alma de cualquier maquinaria o aparato electrónico diseñado para cubrir necesidades específicas es la programación que lleve en los procesadores para poder funcionar de forma correcta y cumplir con las expectativas del cliente o usuario, a esto se le llama **software** o **sistema embebido**. Un Sistema embebido se puede definir como un sistema diseñado y programado para realizar tareas o funciones específicas, a diferencia de los ordenadores de propósito general (como lo son las computadoras personales) que se diseñan para cubrir una amplia gama de tareas o necesidades.

Un ejemplo de un sistema embebido es una tarjeta UNO de Arduino como la mostrada en la Ilustración 1. El hardware de dicha tarjeta está diseñado para cubrir las necesidades para las cuales sea programada.

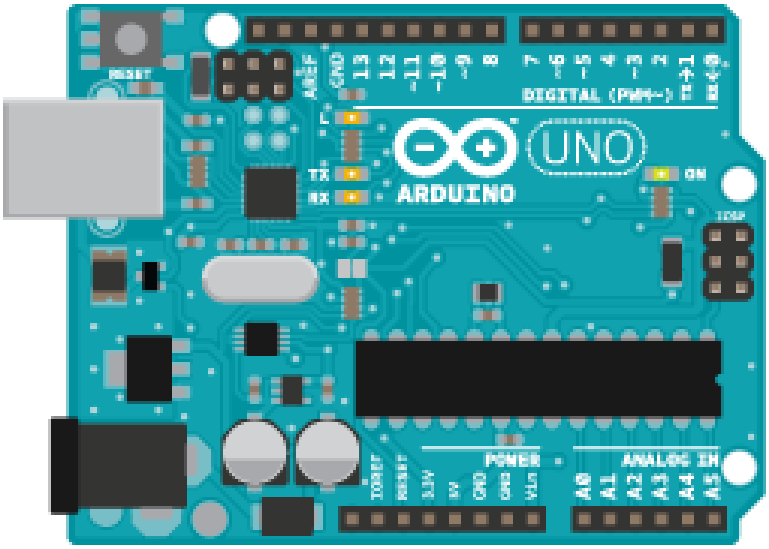

Ilustración 1 Arduino UNO

En el caso de la aviación, el sistema embebido usado dentro de los motores se denomina FADEC (*Full Authority Digital Electronic Control*), el cual es un hardware certificado para poder usarse dentro de ambientes extremos en el motor del avión.

El FADEC, contiene una computadora digital llamada EEC (*Electronic Engine Controller*) la cual mantiene el control de la potencia requerida por la aeronave por medio de actuadores como válvulas y servo motores para el mejor desempeño del avión. La Ilustración 2 representa un FADEC de un motor de avión.

Ilustración 2 FADEC usado en motores de avión

Por seguridad, los FADECs cuentan con 2 canales (Ilustración 3), ambos canales procesan la misma información, esto para que en caso de que alguno de los canales falle, el canal de respaldo pueda suministrar información al FADEC, esta redundancia no solo ocurre en los canales sino también en muchos de los sensores más críticos del avión como lo son sensor de altitud, velocidad y posicionamiento.

Ilustración 3 Canales del FADEC

Los sistemas embebidos se pueden programar en varios lenguajes dependiendo la aplicación, por ejemplo, si el tiempo de respuesta del sistema no es de alta importancia se puede usar JAVA, en caso contrario en el que se necesite una respuesta del sistema rápida, se pueden usar los siguientes lenguajes:

- Lenguaje ensamblador
- C
- C++
- Programación a bloques

Dentro del ramo de la aviación, la programación a bloques es la más usada, ya que existen softwares o plataformas de programación que están certificadas y calificadas para poder ser usadas en sistemas críticos como los usados en las aeronaves o automóviles.

Entre las plataformas más comunes para la programación a bloques se encuentra Simulink de MATLAB y SCADE de ANSYS®. De estas dos, la principal o más usada es SCADE Suite® ya que se ha aprobado por las agencias regulatorias de la aviación, dichas agencias regulatorias se explican más adelante.

SCADE Suite® es un producto de ANSYS® que permite a los programadores poder desarrollar aplicaciones críticas para verificar, simular y generar código certificable o calificado a base de bloques (*Model-Bases Development*).

La programación basada en bloques es un método visual y matemático de solución de problemas o algoritmos relacionados con control, procesamiento de señales y comunicación de sistemas. Es usada en la industria aeroespacial, automotriz, y en aplicaciones de de control de crítico.

Las ventajas de la programación con base a bloques son las siguientes:

- Es bastante sencilla de aprender y muy intuitiva, por lo que la curva de aprendizaje es corta.
- Los errores son fáciles de localizar
- Los diseños son fáciles de reutilizar y modificar

La gran desventaja de la programación basada en bloques es que existen bloques limitados, por lo que los desarrolladores deben ingeniárselas para programar con los componentes existentes, además de que un diseño podría requerir un gran número de pruebas y error para poder asegurar que será a prueba de fallas.

Un sistema embebido cubre las siguientes características genérales:

- Tiene un número de funciones o tareas predefinidas
- Fuente de alimentación limitada
- Funcionamiento de tiempo real
- Periféricos e interfaces limitados
- Recursos de memoria limitados

Entre los periféricos o interfaces que un sistema embebido usa, se listan los siguientes:

- Interfaces Hombre-Máquina
- Teclados
- Monitores
- Interruptores o botones
- Interfaces con otros componentes o dispositivos
- I2C
- SPI
- ISA
- Periféricos
- Ethernet
- RS232
- UART

2.2. FACTORES NECESARIOS PARA LA CERTIFICACIÓN DE SOFTWARE EMBEBIDO

La diferencia entre los sistemas embebidos comunes y los usados en la aviación son las normas que los rigen y que tienen que cumplir, ya que, para poder salir al mercado, dicho software debe certificarse.

Para lograr la certificación de cualquier software de aviación, se deben seguir normas y estándares preestablecidos, en el caso de la aviación, se deben seguir los lineamientos y normas establecidos por el DO-178B el cual es un estándar de desarrollo de software. Dicho estándar se realizó por la *Radio Technical Commission for Aeronautics* (RTCA).

Dependiendo de la función y propósito del software a desarrollar en la industria de la aviación, se deben asignar niveles de criticidad, puede existir un módulo de software que no cause daño alguno o genere una posible falla al avión, a este se le asignará el nivel más bajo, por otro lado, puede existir un módulo de software el cual sea crítico y pueda provocar, en caso de falla, pérdida de vidas humanas, a este módulo se le asignará el nivel más alto de criticidad.

Los niveles de criticidad se dividen de acuerdo al DO-178B como se muestra en la Tabla 1.

Tabla 1 Criticidad de software en aviación

Nivel	**Condición de falla**	**Posible resultado**
A	Catastrófica	La falla puede causar un choque. Ocasiona perdida de funciones criticas requeridas para volar de manera segura.
B	Dañina	Tiene un gran impacto en la seguridad o desempeño. Reduce las posibilidades de los pilotos para operar el avión con seguridad o puede causar daños a los tripulantes.
C	Mayor	Falla significativa, tiene menos impacto que la dañina. Puede ocasionar molestias a los pilotos.
D	Menor	La falla no es perceptible. Puede causar un cambio de ruta de vuelo de rutina.
E	Sin efecto	No tiene impacto en la seguridad o en la operación del avión

Aparte de establecer estos niveles, el DO-178B es una guía con tareas definidas y con objetivos a cumplir para asegurar la calidad del software, para esto se necesitan documentar las etapas del proceso de desarrollo del software. A continuación, se nombran los documentos requeridos para cada etapa de un proyecto de software.

2.2.1. Planeación

No hay documentos requeridos para esta etapa del desarrollo de software de acuerdo con el DO-178B.

La cantidad de documentos también va relacionada a el nivel de software que se esté desarrollando, en este caso, como el software a desarrollar no interviene con las funcionalidades de la aeronave, se considera como nivel D o inferior. Para dicho nivel de software no se requieren documentos de aseguramiento de la calidad.

2.2.2. Desarrollo

El DO-178B especifica el rigor y las tareas a completar para asegurar la calidad del software a desarrollar, los documentos requeridos para esta etapa son los siguientes:

- *Software requirements data* (SRD): Contiene los requerimientos de alto nivel necesarios para el desarrollo y codificación del software.
- *Software design description* (SDD): Contiene prácticas que se deben seguir para que los diseños tengan la calidad necesaria y cumplan con las regulaciones.
- Trazabilidad de requerimientos al código desarrollado

2.2.3. Verificación

Para esta etapa, el DO-178B requiere que exista trazabilidad entre las pruebas realizadas y los requerimientos a probar. Los documentos requeridos son los siguientes:

- *Software verification cases and procedures* (SVCP) contiene instrucciones para la verificación del software.
- *Software verification results* (SVR), este documento contiene los siguientes puntos:
    - Revisión de requerimientos y código
    - Casos de prueba del código objeto
    - Análisis de porcentaje de código probado

2.2.4. Configuration management

Con el objetivo de llevar un control de versiones de software y de documentos, es necesario tener implementado un proceso de *configuration managment*, el cual pueda llevar el control de reportes de problemas, cambios en el software y pruebas realizadas entre otros. Los documentos que esta etapa requiere son los siguientes:

- *Software configuration index* (SCI) es el documento que contiene número de versión e instrucciones de carga del software.
- *Software life cycle environment configuration index* (SECI) contiene las versiones del software usado para el desarrollo del proyecto.

2.2.5. Aseguramiento de la calidad

El proceso de aseguramiento de la calidad según el DO-178B, conlleva la realización de los siguientes procedimientos:

- *Software quality assurance records* (SQAR): documentar el proceso de pruebas y procedimientos seguidos para asegurar la calidad del entregable.
- *Software conformity review* (SCR): revisión del software a entregar conforme a requerimientos.
- *Software accomplishment summary* (SAS): documentar versiones, mejoras y cosas nuevas que se agregaron al software en esta versión.

2.2.6. Metodologías de desarrollo de software

Al comenzar a desarrollar un proyecto de software, lo primero que se debe decidir es el tipo de metodología a seguir, esto conllevara a muchas otras decisiones como el tipo de pruebas o el tipo de *configuration management* a usar.

Existen diferentes tipos de metodologías de software, escoger una depende del tipo de software, el ambiente de desarrollo, el tamaño del equipo, la criticidad del software, etc.

Podemos separar las metodologías de desarrollo de software en 2 categorías, metodologías ágiles y metodologías tradicionales, la diferencia entre ambas se explica en la Tabla 2.

Tabla 2 Metodologías de desarrollo de software

Metodologías Agiles	**Metodologías Tradicionales**
Se basan en heurísticas provenientes de prácticas de producción de código	Se basan en normas provenientes de estándares seguidos por el entorno de desarrollo
Preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Proceso menos controlado	Proceso muy controlado, numerosas normas
Grupos pequeños (<10)	Grupos grandes
Pocos artefactos	Muchos artefactos
Menor énfasis en la arquitectura del software	La arquitectura del software es esencial

En la industria de la aviación se usan ambas metodologías, pero las metodologías tradicionales tienen mayor auge debido a que el proceso debe ser controlado y deben llevarse a cabo revisiones minuciosas para asegurar la calidad y seguridad del software.

Entre dichas metodologías tradicionales se pueden destacar las siguientes:

- Cascada
- Espiral
- Incremental

2.2.7. Método de cascada

El método de cascada es un proceso secuencial de actividades o etapas que van desde la definición de requerimientos hasta la verificación y mantenimiento del producto.

Las estepas de esta metodología de desarrollo se pueden ver en la Ilustración 4.

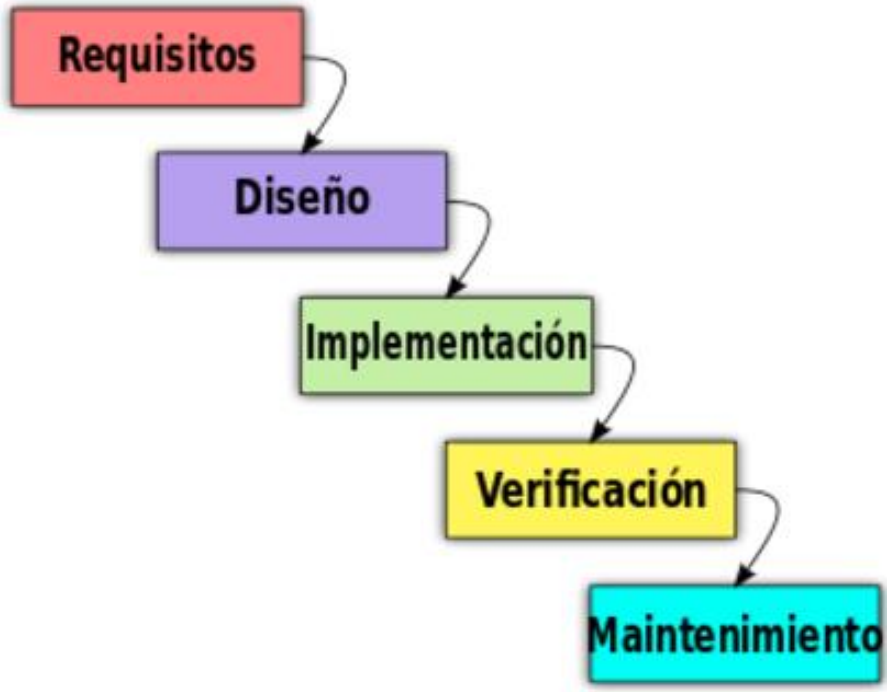

Ilustración 4 Metodología de Cascada

La etapa requisitos o análisis de requerimientos consiste en poner por escrito las necesidades del cliente o producto.

El diseño es la arquitectura o estructura interna del producto o software, dicha estructura normalmente se representa por medio de diagramas a bloques.

La implementación es la programación o codificación del software a partir de los requisitos definidos.

La etapa de la verificación, la cual es de las más tardadas e importantes para la calidad del software, son todas las pruebas necesarias para asegurar que el producto está haciendo las tareas para las que fue diseñado y asegurar o minimizar la presencia de errores.

La última fase del método de cascada es la etapa de mantenimiento, la cual tiene como objetivo optimizar o arreglar el software en caso de ser necesario.

2.2.8. Método de espiral

La metodología de espiral se basa en actividades que conforman una espiral y cada bucle o iteración de dicha espiral son un conjunto de actividades que en cada iteración se repiten hasta lograr el producto final.

Para cada ciclo hay cuatro actividades:

- Determinar Objetivos (Obtener requisitos)
- Análisis del riesgo (Análisis de posibles eventos no deseados)
- Desarrollar y probar
- Planificación

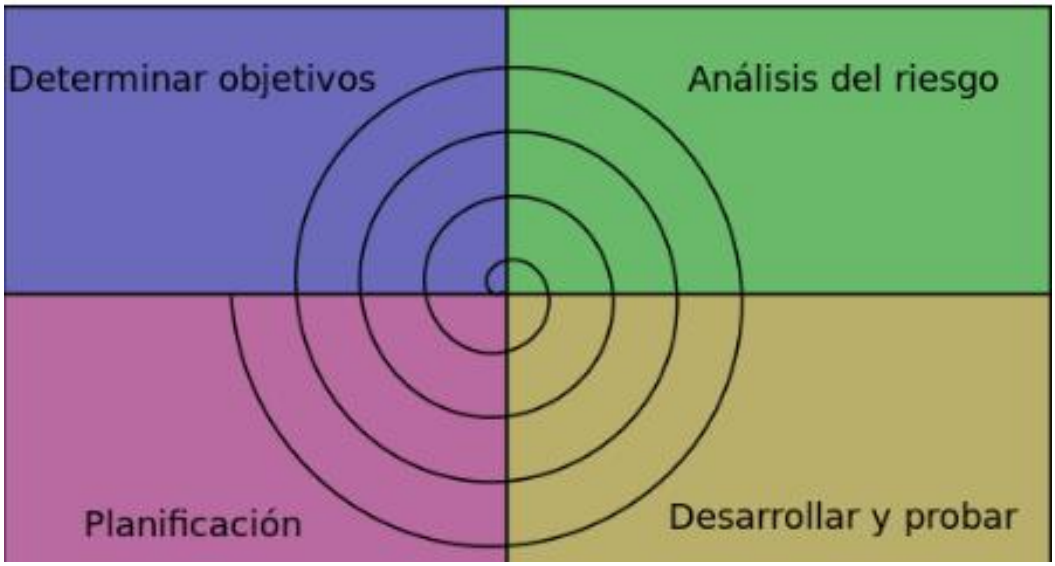

Ilustración 5 Metodología espiral

2.2.9. Metodología incremental

La metodología incremental, también llamada cascada iterativa, es de las más usadas en la industria de la aviación dado que se pueden ver avances y entregar prototipos de forma relativamente rápida. Combina los elementos del modelo de cascada con una filosofía de construcción de prototipos, lo cual hace a esta metodología una de las más eficientes para mostrar resultados al cliente en etapas tempranas del desarrollo de un proyecto.

Entre las ventajas de esta metodología se encuentras las siguientes:

- Se genera un software funcional en etapas tempranas del ciclo de vida del software
- Reduce costos por cambios en el alcance del proyecto ya que es una metodología flexible
- Las pruebas y la depuración son más sencillas ya que se hacen sobre partes pequeñas o módulos

Los inconvenientes que puede llegar a tener esta metodología son los siguientes:

- Se requiere mucha experiencia para poder definir el alcance de cada incremento
- Cada fase de la iteración no debe superponerse con otra
- Debido a que se implementa el diseño en etapas tempranas del desarrollo del proyecto, puede haber incongruencias con los requerimientos de la arquitectura

Para una mejor explicación de este tipo de metodología, la Ilustración 6 muestra las etapas e iteraciones que se llevan a cabo en la metodología incremental.

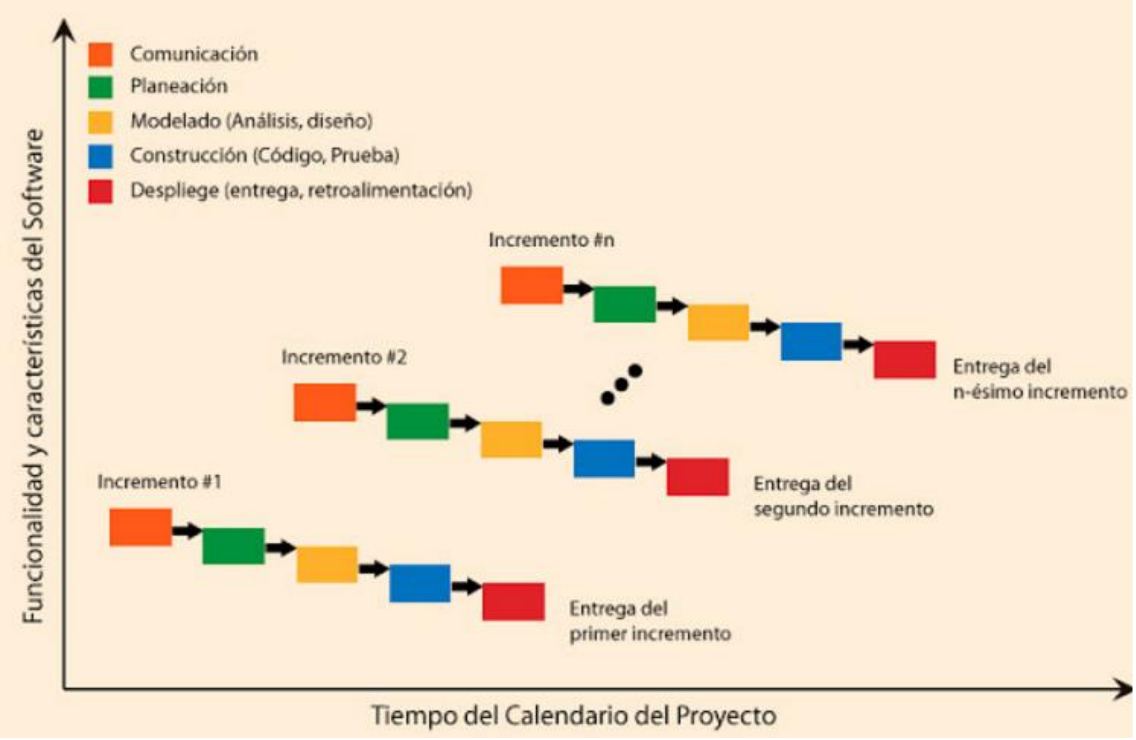

Ilustración 6 Metodología Iterativa

2.2.10. Tipos de pruebas de software

Existen diversos tipos de pruebas para el software las cuales tienen la finalidad de asegurar la calidad y correcto funcionamiento de dicho software, estas pruebas de categorizan como sigue:

- Pruebas estáticas
    - Inspecciones
    - Pruebas de escritorio
    - *Code review*
- Pruebas dinámicas
    - *White Box*
    - *Black Box*
    - Prueba unitaria
    - Prueba de integración
    - Pruebas del sistema
    - Pruebas de regresión
    - Pruebas de aceptación

A continuación, se describe cada una de dichas pruebas.

2.2.11. Pruebas estáticas - Inspección

El objetivo de este tipo de pruebas es detectar e identificar defectos en el software mediante una inspección del código línea a línea, dicha inspección se hace por medio de expertos en la materia que no hayan participado en el desarrollo del código a revisar.

El resultado de este tipo de pruebas es remover defectos en el código y malas prácticas usadas por el desarrollador, así como verificar que el código cumpla con los estándares establecidos por la compañía de software.

2.2.12. Pruebas estáticas - Pruebas de escritorio

Este tipo de pruebas consiste en hacer una corrida del código a mano con valores reales para verificar si el código tendrá el comportamiento deseado y cumple con los requerimientos del cliente.

El resultado de este tipo de prueba es remover defectos en el código por una mala codificación o un error por parte del desarrollador.

2.2.13. Pruebas estáticas - Code Review

Este tipo de prueba es parecida a la inspección, con la diferencia de que la inspección verifica el código, el formato y estándares, y el *code review* se enfoca en errores de codificación mediante la revisión línea por línea del software.

El resultado de este tipo de prueba es remover defectos en la programación del software para evitar errores en el producto.

2.2.14. Pruebas dinámicas - White Box

Este tipo de pruebas se realizan sobre el código en ejecución, se proveen entradas y se ve todo el procesamiento de las entradas hasta ver el resultado final en las salidas.

Con este tipo de pruebas se pueden encontrar errores en cualquier parte y decisión del software, aunque es una prueba tardada y extensiva, asegura el correcto funcionamiento del software y la calidad del mismo.

Entre las técnicas de las pruebas de caja blanca más conocidas tenemos la cobertura, que consiste en la verificación de que todos los caminos lógicos del software son alcanzables en función de los diferentes valores de entradas de los parámetros.

Otra técnica bastante conocida es la *Mutation Testing.* Se basa principalmente en realizar ligeras modificaciones en el programa para dar lugar a un comportamiento anómalo del mismo (resultados distintos) y verificar si la estrategia de *testing* utilizada es capaz de detectar estos cambios.

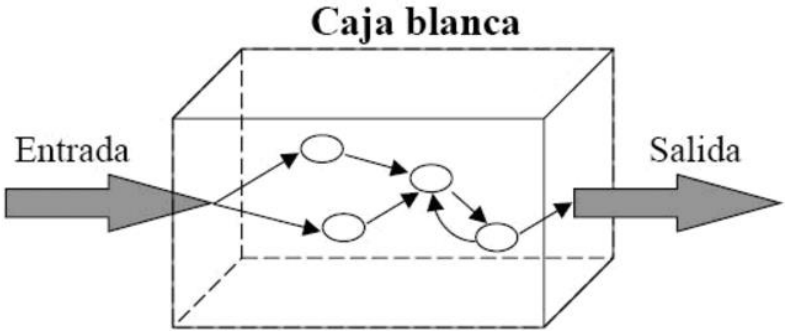

Ilustración 7 Pruebas de caja blanca

2.2.15. Pruebas dinámicas - Black Box

Las pruebas de *black box* o caja negra, se basan en el entendido de que a ciertas entradas corresponde una salida en específico, consiste en inyectar diferentes valores a la entrada y ver el resultado a la salida.

Por tanto, la diferencia fundamental respecto a las pruebas de caja blanca es que en este caso no se trabaja con el código fuente sino con el programa

Ilustración 8 Pruebas de caja negra

2.2.16. Pruebas dinámicas - Prueba Unitaria

Este tipo de pruebas se basan en separar el código en partes pequeñas y probar cada una de esas partes de forma independiente. Ayuda a localizar errores en partes especificas del código, pero dada la cantidad de casos de prueba que requiere, es tardada.

Para realizar este tipo de pruebas se pueden usar distintas herramientas que ayudan a realizar, automatizar y reutilizar los casos de prueba dependiendo del ambiente del software a probar como lo son:

- *JUnit*: Entorno de pruebas para Java
- *PHPUnit*: Sistema para la realización pruebas unitarias en PHP
- *CPPUnit*: Versión del *framework* para lenguajes C/C++
- *NUnit*: Versión del *framework* para la plataforma
- Test: Entorno desarrollado por GE para realizar pruebas unitarias con las especificaciones necesarias para la certificación del software *GE Unit*

2.2.17. Pruebas dinámicas - Prueba de Integración

Este tipo de pruebas se realizan después del proceso de pruebas unitarias y conlleva probar que todas las unidades funcionen bien en conjunto. Se centra en probar que los componentes unitarios se comuniquen de forma adecuada unos con otros, ya sean componentes de hardware y/o software.

2.2.18. Pruebas dinámicas - Pruebas del Sistema

Este tipo de pruebas tiene como objetivo encontrar fallas o discrepancias entre los requerimientos y el diseño del software, en otras palabras, se enfoca en probar como el software resuelve la necesidad que se especifica en los requerimientos.

2.2.19. Pruebas dinámicas - Pruebas de regresión

Las pruebas de regresión se realizan una vez que se generó un cambio en el software ya sea por un error o alguna modificación. Se evalúa que la parte de software que no se modificó siga funcionando de forma correcta y que los cambios realizados están correctamente implementados.

2.2.20. Pruebas dinámicas - Pruebas de aceptación

Es el último paso de la fase de pruebas para asegurar la calidad del software. Una vez que cada módulo se probó de forma separada y que la función en conjunto es correcta, se realizan las pruebas de aceptación en conjunto con el usuario final del producto.

En el caso de la aviación, este tipo de pruebas se llaman *flight test*, en el que el software se carga al motor y/o componentes del avión y se llevan a cabo varios vuelos de prueba, posterior a los vuelos, se analizan los datos y parámetros para confirmar que todo funciona de acuerdo a requerimientos y que ninguna falla no esperada surgiera durante el vuelo.

2.2.21. Importancia de la certificación de software en la industria de la aviación

El software en la industria de la aviación se considera, en su mayoría, software de categoría A, ya que vidas humanas están en juego en caso de una mala programación.

Por esto, la certificación del software tiene una gran importancia, ya que, al certificar el software, se está afirmando que se hicieron todas las pruebas necesarias en todos los posibles escenarios para con ello dar por sentado que no ocurrirá una falla que conlleve a la posible pérdida de vidas humanas.

En el caso de este documento, se explora la automatización y reducción de tiempos en la certificación de software nivel E el cual, si falla, no ocasionara nada mayor a una posible alerta visual.

Aunque el software nivel E no conlleve tanta severidad, la certificación es necesaria e importante, ya que este tipo de nivel de software, involucra en gran medida, la recolección de datos de vuelo en las diferentes etapas de vuelo del avión, dichas etapas se explican más adelante.

Los datos de vuelo recolectados se utilizan para el análisis de un posible mantenimiento preventivo al motor y con esto agendar el mantenimiento con anticipación para evitar retrasos por fallas o por alertas existentes.

2.2.22. Definición y Utilidad de los Reportes de Vuelo

Un Reporte de Vuelo es una captura de información de un vuelo en un tiempo determinado, el cual contiene datos con respecto a todos los sensores que tiene la aeronave tanto en el motor como en el fuselaje.

Dichos reportes se generan y guardan dentro de la computadora del avión, comúnmente llamada FADEC.

Los reportes de vuelo en la actualidad son de gran importancia en la industria de la aviación para realizar análisis estadísticos de los datos capturados con el fin de tratar de evitar daños y surgimiento de fallas en la aeronave, dando como resultado el tener una operación de vuelo de alta calidad. Además, funciona como complemento a las grabaciones realizadas por una caja negra.

Los reportes de vuelo son comúnmente archivos de extensión "csv" (*Comma separated values*), este tipo de archivos son documentos en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea. Por estas características, si se tienen datos que contengan

una coma, un salto de línea o una comilla doble deben ser encerrados entre comillas. La ventaja de este tipo de archivos es que, por su formato, ocupan un espacio en disco relativamente pequeño dependiendo de la información que contenga el archivo, es por esto que son usados en la industria de la aviación, donde el tamaño de los archivos deriva en espacio de almacenamiento y en costo de transferencia de dichos archivos, los cuales dependiendo del tamaño o del tiempo del vuelo, pueden ser partidos en varios archivos tipo csv.

Existen en promedio 11,000 aviones en el aire cada minuto del día, por lo que el optimizar el espacio de estos reportes es muy importante, el tamaño de estos archivos es también esencial para el análisis de los datos.

Para realizar el análisis de los reportes, estos se deben transmitir bajo cierto protocolo para asegurar que no se corrompan los datos y lleguen de forma segura a su destino. Por convención de las agencias regulatorias mundiales, en aviación se usa el protocolo ACARS (*Aircraft Communications Addressing and Reporting System*).

ACARS es una red de comunicaciones aire/tierra; que se usa para transmitir o recibir datos de forma automática o manual. Está dedicado a mantenimiento y operaciones comerciales. Esto hace posible que una línea aérea pueda comunicarse con las aeronaves de su flota en la misma forma en que es posible intercambiar datos a través de una red digital terrestre. ACARS usa el identificador único de la aeronave y el sistema tiene algunas características que son similares a las utilizadas actualmente para el correo electrónico. Las comunicaciones ACARS se dirigen automáticamente a través de una serie de bases en tierra ARINC (*Aeronautical Radio Inc*.) y computadoras para el operador de la aeronave.

Lo anterior permite la comunicación sin intervención humana entre el avión y la compañía aérea, liberando a la tripulación de operaciones repetitivas. Los mensajes habituales son:

- Carga de pasajeros
- Informes de despegues
- Informes de aterrizaje
- Combustible
- Datos de las características del motor

Para llevar a cabo esta comunicación, existen en la actualidad cuatro redes principales en el mundo:

- ARINC en USA
- CANADIAN en Canadá
- JAPANESE en Japón
- SITA en las otras regiones

La información recabada puede ser solicitada por la empresa y se recuperan de la aeronave en intervalos periódicos o a demanda. Antes de ACARS este tipo de información habría sido transferida a través de VHF (*Very High Frequency*).

Otra información puede incluir planes de vuelo, información meteorológica significativa (SIGMET), listas de la tripulación, declaraciones de carga, servicio automático de información terminal (ATIS), informes en la ruta y el tiempo de destino, autorizaciones e informes de combustible.

Los componentes del ACARS incluyen una Unidad de Gestión de las Comunicaciones (CMU); actúa como un *router* para todos los datos transmitidos o recibidos del exterior, que se ocupa de la recepción y transmisión de mensajes a través del transceptor de radio VHF y la unidad de control que proporciona la interfaz de la tripulación y se compone de una pantalla y una impresora.

La Red de tierra ACARS comprende estaciones remotas de transmisión/recepción y una red de computadoras que analizan y envían reportes a las autoridades correspondientes.

Hay dos tipos de mensajes ACARS, los mensajes descendentes (*downlink*) que se originan en la aeronave y los mensajes de enlace ascendente (*uplink*) que se originan en las estaciones terrestres. La velocidad de datos es baja y los mensajes se conforman de caracteres alfanuméricos.

La CDU (Control de Unidades de Visualización, por sus siglas en inglés) es una combinación de pantalla y un teclado, y es la interfaz principal del piloto con el sistema ACARS.

2.2.23. Fases de vuelo para recolección de datos

Los reportes ACARS se generan de acuerdo a las necesidades del cliente o las recomendaciones del fabricante, por ejemplo, se pueden tomar datos en diferentes puntos de la etapa de arranque, así como en la fase de despegue y aterrizaje.

Las fases de vuelo de un avión, en lo general, se pueden ver en la Ilustración 9.

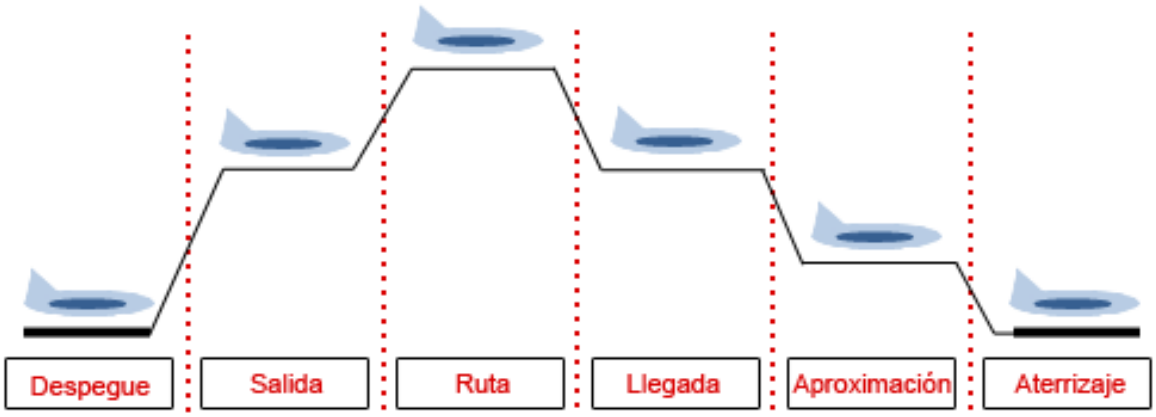

Ilustración 9 Fases de vuelo

A lo largo de todas estas fases de vuelo, se pueden tomar o recolectar datos que, al analizarlos, darán como resultado la salud del motor, por ejemplo, se pueden tomar mediciones del nivel de aceite en las etapas de despegue, aterrizaje y de crucero para con base en estos datos determinar que etapa consume más aceite. Con estos datos se podría llegar a determinar la salud de los filtros de aceite del motor o la etapa de vuelo en que el motor lleva más desgaste.

Todos estos datos se analizan en tierra mediante analíticos los cuales hacen tendencias con base a los datos de diferentes motores para determinar si necesitan programarse para mantenimiento o no.

2.3. EVOLUCIÓN DEL MANTENIMIENTO EN LA AVIACIÓN

Desde los inicios de la aviación, la seguridad y el mantenimiento siempre han jugado un papel muy importante, ya que vidas humanas están en riesgo en caso de algún incidente. Debido a esto se tuvo la necesidad de agregar diversos tipos de sensores para monitorear variables importantes para el funcionamiento correcto de una aeronave.

En un principio, fue suficiente con desplegar valores de dichas variables en el tablero del piloto, conforme paso el tiempo, más preocupaciones surgieron, como lo son el alto costo y tiempo del mantenimiento, cancelaciones de vuelos por fallas y alto costo de refacciones innecesarias.

Actualmente es ampliamente aceptado que la aviación comercial es la forma más segura para viajar. Hoy en día, las aerolíneas comerciales sufren menos de dos accidentes por millón de despegues.

A principios de los 60s, los accidentes aéreos ascendían a 60 accidentes por cada millón de despegues, en la actualidad esto equivaldría a 2 accidentes aéreos diarios. Esto fue debido a que en ese entonces el mantenimiento era sinónimo de reparaciones periódicas o hasta que algo dejara de funcionar.

Debido a tantos accidentes en esa época, el mantenimiento en la aviación evoluciono de tal forma que ahora es un proceso analítico y sistemático el cual usa los reportes de vuelo para generar tendencias y analizar comportamientos. Con esto se puede predecir con una gran confianza, el momento en el que cierta parte del avión necesitará mantenimiento.

Dichas tendencias se hacen por medio de análisis estadísticos teniendo datos históricos de muchos vuelos, algunos otros solo se enfocan a que una variable no pase cierto rango, por ejemplo, que la temperatura del FADEC no supere los 100°C por dar un ejemplo.

Gracias a estas tendencias y análisis de los datos, el mantenimiento de las partes del avión se ha reducido en gran medida, esto dependiendo de la parte y del uso del avión. Esto ha conllevado en una reducción importante en costos de mantenimiento y un aumento en la disposición de las aeronaves.

3. PROCEDIMIENTO DE INVESTIGACIÓN

El análisis del proceso de pruebas formales en la industria de la aviación es largo y conlleva varias fases de prueba, dependiendo la etapa del desarrollo es la prueba que se aplica al software. dichas fases se listan a continuación:

- Simulación en PC
- Target Hardware Test
- Test Case Review
- Pruebas de Sistema
- Pruebas Formales
- Pruebas de integración

3.1. SIMULACIÓN EN PC

La simulación en PC se realiza en la etapa de desarrollo del software y tiene el objetivo de eliminar cualquier probable error al probar todos los posibles escenarios del software.

Dichas pruebas se realizan por medio de un simulador que emula el comportamiento del hardware real en un entorno seguro.

3.2. TARGET HARDWARE TEST

Las pruebas en el hardware se realizan con el objetivo de asegurar que el software se comportará de la forma que se comportó en la simulación en PC, pero en el hardware real.

Dichas pruebas se corren sobre el hardware diseñado para ejecutar el software, pero en un laboratorio con las conexiones mínimas requeridas para llevar a cabo las pruebas.

3.3. TEST CASE REVIEW

El proceso de *test case review* consiste en la revisión de los resultados de correr las pruebas en el hardware real, dichos resultados son revisados por una persona ajena a la lógica y se realiza la revisión contra los requerimientos del software.

3.4. PRUEBAS DE SISTEMA

Parecidas a las pruebas en el hardware real, pero con la diferencia de que se conectan todos los periféricos y se verifica la interacción entre cada uno de ellos.

Tiene el objetivo de encontrar posibles fallas con la comunicación entre distintos componentes.

3.5. PRUEBAS FORMALES

Las pruebas formales son parecidas a las pruebas de sistema, con la diferencia de que las pruebas formales se realizan sobre la versión del software que se entregara al cliente, a estas pruebas se les denomina pruebas de integración.

3.6. PRUEBAS DE INTEGRACIÓN

A las pruebas finales que se realizan junto con el cliente en sus instalaciones, con el objetivo de que el cliente observe el correcto funcionamiento e interacción del software, se les denomina pruebas de integración.

3.7. TIEMPOS Y CUELLOS DE BOTELLA EN PRUEBAS FORMALES

Dentro de este trabajo, se explora a fondo el proceso de pruebas formales, el cual es el más tardado de los diferentes procesos de pruebas para lograr la certificación, esto debido a que el tiempo de espera entre cada prueba es muy extenso y las herramientas tardan mucho en finalizar.

Para llegar a la etapa de pruebas formales, es necesario pasar por todas las fases o etapas del desarrollo sin tomar en cuenta la etapa de mantenimiento, esto quiere decir que se corre el proceso desde los requerimientos y hasta la verificación, dichas fases se pueden consultar en la Ilustración 4.

En la industria de la aviación, después de la etapa de verificación, en el caso de ser el entregable final que se enviara al cliente, se corre el proceso de las pruebas formales del software.

El proceso de pruebas formales conlleva los siguientes pasos o tareas a realizar:

- Paso 1
- Paso 2
- Paso 3
- Paso 4
- Paso 5
- Paso 6

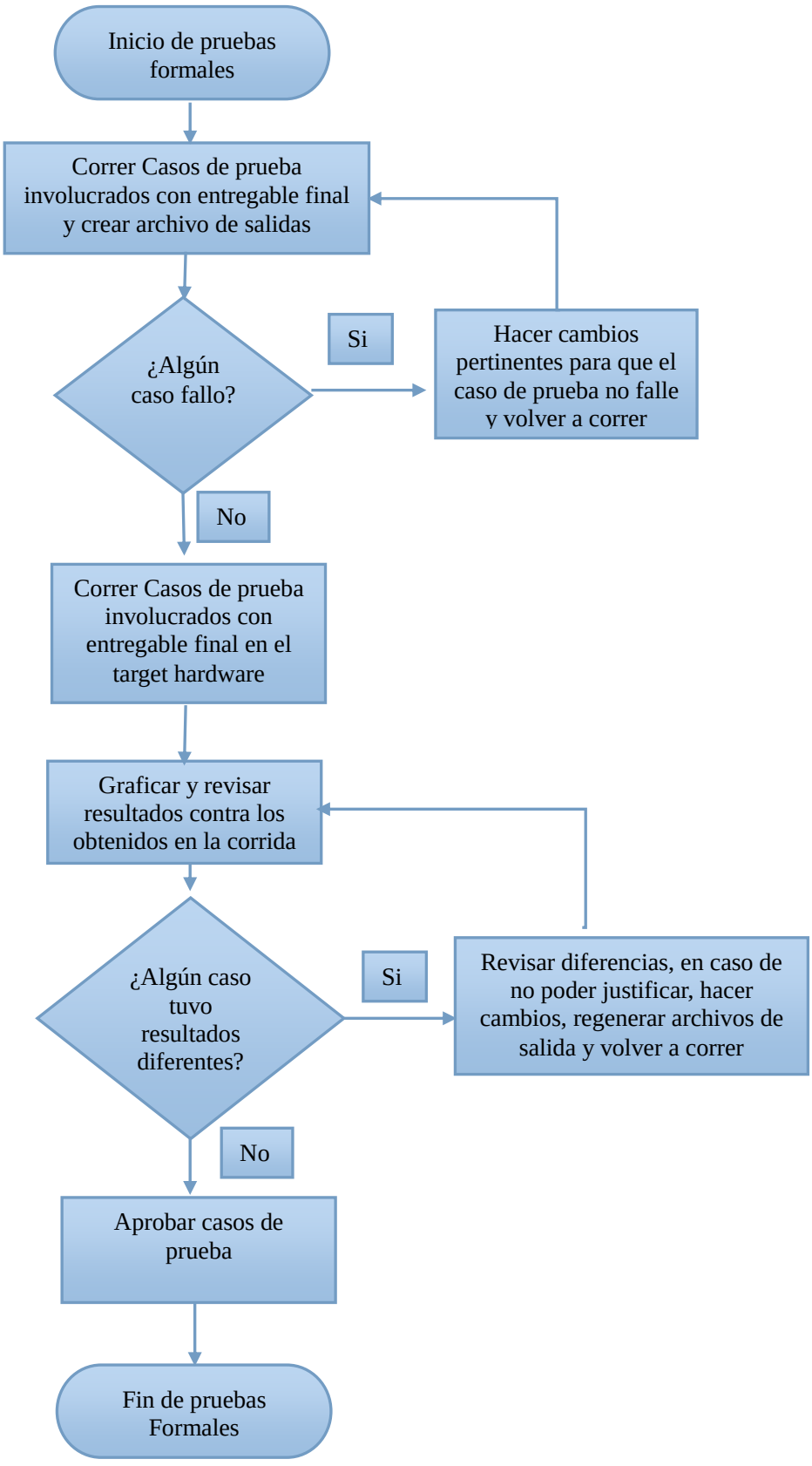

Ilustración 10 Diagrama de flujo de pruebas formales

De los pasos mostrados en el diagrama de la Ilustración 10, los pasos 1 y 5 que es donde se generan los archivos de salida o donde podrían volver a generarse (pasos 1 y 5 respectivamente). Estos pasos son los más tardados de todo el proceso, esto dado que los archivos tardan un promedio de 30 minutos en generarse por cada caso de prueba, dicho tiempo se obtuvo de un estudio realizado sobre el proceso de pruebas formales para tratar de reducir tiempos y encontrar oportunidades de mejora.

Otro punto a tratar con respecto a la generación de archivos de salida es que se tiene que esperar a la generación de archivos de un caso de prueba para comandar la generación de archivos del siguiente caso de prueba.

Uno de los problemas más grandes de este proceso son los tiempos muertos entre cada corrida de casos de uso y entre cada generación y revisión de archivos, ya que la herramienta actual no avisa si algún proceso ya termino y se puede iniciar uno nuevo.

Adicionalmente, el proceso incluye correr los casos de prueba y revisión de sus resultados, dicho proceso toma en promedio 15 minutos por caso de uso.

Para ciertos programas, hay demasiados casos de prueba por ejecutar, por lo que toma demasiado tiempo generar los archivos de salida. A continuación, se explica un ejemplo de pruebas formales para una versión para la compañía de aviación Boeing.

80 casos de prueba necesitaban ser ejecutados, de cada uno se deben generar archivos de salida y correr sus casos de uso.

80 casos con un promedio de 30 minutos entre cada generación de archivos de salida y 15 minutos de la ejecución de los casos de prueba son un total de 60 horas, tomando como referencia histórica un 30% de re trabajo en la generación de estos archivos (paso 5), son otras 18 horas, esto nos da un total de 78 horas.

78 horas representan casi 10 días de una jornada laboral de 8 horas, el tiempo en la generación de los archivos es difícil de ser acortado. Por dicho motivo, se requiere una forma más sencilla de optimizar el proceso a un bajo costo.

Es por estos motivos que 3 personas deben estar en el proceso, dado que es demasiado arriesgado involucrar a una o 2 personas, ya que solo se tienen en promedio 2 semanas para completar el proceso, y los 10 días que se tarda en terminar la de generación de archivos, no incluyen el tiempo de revisión y aprobación de los casos de uso.

3.8. SOLUCIONES PROPUESTAS

Para lograr hacer más eficiente el proceso de pruebas formales para certificación de software de aviación, se tomaron en cuenta varios factores, los cuales se mencionan a continuación:

- Cada cuanto se realizan dichas pruebas: Al menos 4 veces por año.
- Cuantos casos de prueba en promedio se corren en las pruebas formales: 55 casos de prueba.
- Cuantos miembros del equipo en promedio participan en el esfuerzo de pruebas formales: 3 miembros.
- Cuánto tiempo se tiene para terminar el proceso de pruebas formales: 2 semanas.

Dichos factores llevaron a la conclusión de que una herramienta que ayude a hacer más rápido el proceso, tendría un costo-beneficio de alto impacto para la empresa, ya que, en lugar de 3 personas, una herramienta permitiría que una sola persona estuviera en el proceso y terminara de una forma más rápida y eficiente.

Con el objetivo de obtener la mejor solución, se llevó a cabo una lluvia de ideas, en la que al final resaltaron 3 ideas que ayudarían a hacer más rápido el proceso de pruebas formales.

3.8.1. Idea 1: Modificar la herramienta que genera los archivos de salida para tratar de hacerla más eficiente.

La idea de modificar la herramienta generadora de los archivos de salida con el objetivo de intentar agilizar el proceso, podría conllevar mucho tiempo de desarrollo, ya que esa herramienta no se realizó en México, lo cual implica que, permisos de exportación del software serían necesarios para que el equipo en México pudiera acceder al código de la herramienta, dichos permisos podrían ser costosos y tardados.

Modificar la herramienta, aunque podría ayudar, no existe certeza de cuanto se podría optimizar ni de cuánto podría ayudar al proceso.

Dicha herramienta recauda información de distintas bases de datos en donde el software se encuentra y por medio de un algoritmo analiza las entradas, salidas y lógica a la que cada caso de uso esté ligado.

El resultado de la herramienta son varios archivos que contienen información sobre si todos los posibles caminos lógicos del software se están cubriendo y sobre si todas las entradas y salidas se están ejercitando dentro del caso de uso.

Para que el caso de uso sea aprobado, todas las entradas, salidas y posibles caminos lógicos deben aparecer como cubiertos o justificados en los archivos de salida, lo cual implica que, si no se está cubriendo alguno de los puntos anteriores, entonces el caso de uso debe ser modificado y la herramienta debe volver a ser ejecutada para generar nuevos archivos de salida mostrando todos los caminos lógicos cubiertos.

Otro punto a considerar es que los comandos necesarios para ejecutar la herramienta son difíciles de recordar. Dichos comandos deben llevar los siguientes datos:

- Comando para generar archivos -nombre de caso de prueba -Nombre del archivo de software a revisar
- Comando para revisar archivos -nombre de caso de prueba -Nombre del archivo de software a revisar

En conclusión, modificar la herramienta que genera los archivos de salida, aunque prometedor, no es una opción viable ya que podría no cambiar mucho el tiempo en que los archivos de salida son generados y podría llevar mucho tiempo el tramitar los permisos de exportación de dicho software.

3.8.2. Idea 2: Herramienta que revise los archivos de salida en el paso 5 y en caso de encontrarse un error, volverlos a generar.

El paso 5 del proceso de pruebas formales incluye la revisión de los resultados de los casos de uso (archivos de salida y gráficas del caso de uso), en dado caso de que los resultados sean diferentes, se tienen que hacer cambios y volver a generar los archivos. Este re-trabajo, acorde a datos históricos, toma alrededor de un 30% adicional del tiempo de ejecución.

Una herramienta que revise los resultados de forma automática ayudaría a reducir este 30% de tiempo adicional y ayudaría a agilizar o automatizar el proceso.

Un contratiempo de esta idea es que al hacer una herramienta que revise los archivos, dicha herramienta se tendría que quilificar dado que la FAA y la EASA así lo requieren.

La quilificación de una herramienta es un proceso largo y costoso, dado que el proceso de pruebas formales se realiza unas 4 veces por año, no resulto viable el costo beneficio de llevar a cabo esta idea.

3.8.3. Idea 3: Script que ejecute la herramienta de forma automática

Otra idea es realizar un *script* que nos ayude a ejecutar los comandos necesarios para la herramienta y poder generar y revisar los archivos de salida del proceso de pruebas formales.

Dicho *script* ayudará a que el proceso se pueda ejecutar por una sola persona ya que automatizará los comandos que la herramienta necesita para su ejecución.

Con esto, los tiempos muertos entre la ejecución de un caso de prueba, y la generación de los archivos de salida, se eliminaría por completo.

Esta resulto ser la idea la más sencilla de implementar y con mejor costo beneficio para la compañía por los siguientes motivos:

- Elimina tiempos muertos entre cada ejecución
- El usuario puede ejecutar la herramienta y dedicarse a otras actividades
- La herramienta puede seguir ejecutándose durante la noche

Esto implica que el posible re-trabajo después de analizar los resultados no será cubierto por esta idea, pero esto conlleva a que esta idea no necesita quilificarse debido a que dicho script solo ejecuta la herramienta, pero la revisión final es responsabilidad de un ingeniero.

Aunque esto significa que la herramienta puede necesitar ser corrida más de una vez debido a los posibles errores encontrados por el ingeniero en el paso 5 del proceso, ahorrará tiempo en el proceso debido a la eliminación de lapsos muertos y debido a que puede ser ejecutada por un solo ingeniero en lugar de los 3 ingenieros que en promedio se necesitan para el proceso de pruebas formales.

Por todos los motivos anteriores, se concluyó que esta será la idea a implementarse para optimizar el proceso de pruebas formales.

De entre los posibles lenguajes de programación, se eligió realizar la herramienta en *Perl* debido a que es un lenguaje sencillo de muy alto nivel para automatizar tareas, por otro lado, a que las herramientas que se usan para el proceso de pruebas formales de software también están hechas en este lenguaje de programación.

La forma en que trabajará el *script* será mediante un archivo de entrada, el cual contendrá todo un listado de los casos de uso a correr con su respectivo archivo de software al que está ligado cada caso de, a continuación, se muestra un ejemplo del archivo de entrada.

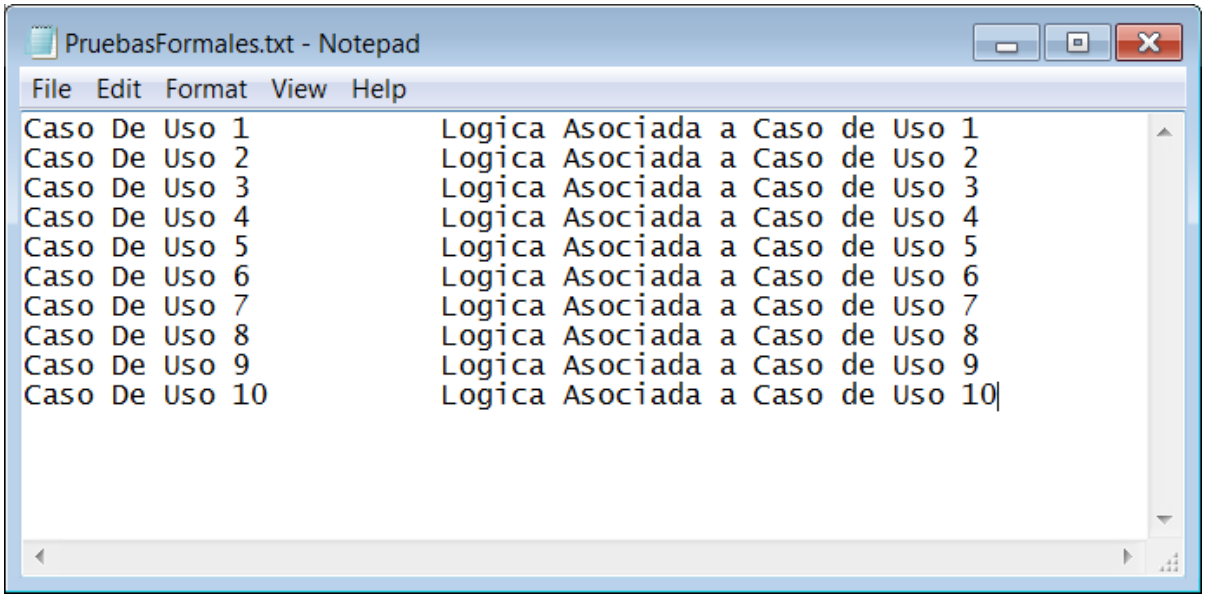

Ilustración 11 Archivo de entrada de script

El *script* aparte de ayudarnos a generar y revisar los archivos de salida, también llevará a cabo el proceso de correr los casos de uso (también incluido en los pasos 1 y 5 del proceso de pruebas formales). Con esto la herramienta ahorrara gran parte del proceso de pruebas formales a la compañía.

Para correr dichos casos de uso se necesita obtener 4 archivos de una base de datos y copiarlos a la ubicación donde se corren los casos de prueba, este proceso también se hará de forma automática, un ejemplo de los 4 archivos necesarios se muestra a continuación:

- NombreDelCasoDeUso.def: Contiene las variables a mover con tiempos definidos
- NombreDelCasoDeUso.int: Contiene las inicializaciones requeridas para el caso de uso
- NombreDelCasoDeUso.opl: Contiene las variables a monitorear durante la prueba
- NombreDelCasoDeUso.gmc: Contiene la información a graficar después de correr el caso de uso

Al terminar de correr el caso de uso, se deben graficar sus resultados para su posterior análisis y comparación (paso 4 del proceso de pruebas formales).

Los pasos que la herramienta deberá seguir se muestran a continuación:

- Paso 1: iterar por cada Caso de uso de la lista:
  - Obtener archivos de cada caso de uso y copiarlos al folder correspondiente.

- Paso 2: Correr cada caso de uso mediante el siguiente comando:
  - runTV NombreDelCasoDeUso

- Paso 3: Generar graficas de los resultados de cada caso de uso mediante el siguiente comando:
  - gplus NombreDelCasoDeUso

- Paso 4: Ejecutar comando para generar archivos de salida de cada caso de uso.

- Paso 5: Ejecutar comando para revisar archivos de salida de cada caso de uso: El proceso de revisión de los archivos implica el poner un nombre de usuario y contraseña de una base de datos restringida, el *script* debe pedir dicho nombre de usuario y contraseña al inicio de la ejecución para ponerlo cada vez que sea requerido.

- Paso 6: Guardar archivos de salida en una ubicación para su posterior revisión por el usuario.

Cada paso conllevaría una interacción del usuario, la cual ya no será necesaria dado que el *script* la realizará de forma automática.

Con esto, el usuario solo deberá realizar la lista de entrada de casos de uso con la lógica asociada correspondiente, dar su nombre de usuario de contraseña solo una vez al inicio y el *script* realizará el resto.

Esto significa que el proceso correrá de forma automática mientras el usuario puede enfocarse en otras actividades.

El código de dicho *script* es información confidencial y propietaria de GE por lo que no se presentará ni explicará en este documento.

4. RESULTADOS

Mediante la implementación de la idea propuesta de un *Script* que ejecute la herramienta de forma automática, se automatizo el proceso de pruebas formales del software de aviación de forma que se redujo el número de personas involucradas en el proceso de 3 a 1 sola persona.

Los resultados de utilizar el *script* son un ahorro de recursos de un 33%, esto al ser usado por una sola persona en lugar de 3, y de una reducción en el tiempo de ejecución del proceso de un 67% aproximadamente, lo que significa que un solo recurso puede terminar el proceso a tiempo dejando días suficientes para la revisión de los entregables, esto tomando en cuenta un promedio de 2 semanas desde que empieza hasta que termina el proceso de pruebas formales.

Se comprobó que, al usar el *script*, una persona puede dejar el proceso corriendo durante la jornada laboral o en la noche, mientras se dedica a otras actividades.

Con esto, el usuario solo debe preocuparse por recabar la documentación generada y revisar los resultados de cada caso de uso, esto con el objetivo de reducir el posible re trabajo y asegurar la calidad de los entregables.

5. CONCLUSIONES

La hipótesis planteada al inicio de este documento que habla sobre una herramienta que reducirá en un 50% el tiempo del proceso de pruebas formales del software de aviación es verdadera, ya que mediante el *script* que automatiza la herramienta generadora de archivos de salida y la simulación de los casos de uso, se logró una reducción del 55% en el tiempo del proceso.

La situación actual de la empresa exige la reducción de tiempos y mejora en la calidad de los entregables, ya que los objetivos de la empresa se alinean hacia una constante mejora en los procesos.

La constante mejora u optimización implica invertir horas en analizar y desarrollar una mejora, antes de realizarla, se debe analizar si la mejora tendrá un impacto alto, ya que nos podríamos encontrar en el caso de que la mejora a realizar cueste mucho más de lo que posiblemente se podría ahorrar con dicha mejora.

Las dos hipótesis restantes se descartan debido a su complejidad y su poca factibilidad para la mejora y reducción de tiempos para proceso de pruebas formales.

6. RECOMENDACIONES

Muchos de los procesos internos de GE pueden ser automatizados con herramientas o *scripts* similares, hay mucha oportunidad de mejora y de optimización en dichos procesos, los cuales deben ser analizados para encontrar la opción más viable para su posible automatización y reducción de tiempos.

Se debe tomar en cuenta que gran parte de las posibles mejoras a los procesos internos de GE puede solo ser vista por las personas que participan en el proceso, por lo que se debe incentivar a las personas a encontrar y proponer mejoras que les ayuden a ser más eficientes día con día y a mejorar la calidad de sus entregables.

Para finalizar, es muy importante tener siempre en mente que la verificación, revisión y aprobación por parte de una persona siempre será necesaria, esto significa que, aunque una herramienta automatice gran parte de un proceso, al final siempre será necesaria la revisión y aprobación de una persona para validar el trabajo de la herramienta.

7. REFERENCIAS BIBLIOGRÁFICAS

Moubray, John (Abril 25, 2017). MANTENIMIENTO CENTRADO EN CONFIABILIDAD (RCM). http://www.mantenimientoplanificado.com. http://www.mantenimientoplanificado.com/art%C3%ADculos_rcm_archivos/RCM2%20EXPLICACION.pdf

J. F. Alonso (marzo 24, 2015). Cuántos aviones vuelan cada día en el mundo. http://abcblogs.abc.es/proxima-estacion/public/post/viajar-aviones-diarios-mundo-16911.asp/

Federal Aviation Authority. Regulations and Guidelines. https://www.faa.gov/

Seguridad Aérea. SISTEMAS TÍPICOS ELECTRÓNICOS/DIGITALES EN AERONAVES. http://www.seguridadaerea.gob.es/media/3785435/modulo05_cap15.pdf

Safety Management (marzo 31 2017). Flight Data Monitoring. http://www.skybrary.aero/index.php/Flight_Data_Monitoring

Safety Management (marzo 31 2017). Flight Data Recorder. http://www.skybrary.aero/index.php/Flight_Data_Recorder_(FDR)

Safety Management (marzo 31 2017). Aircraft Communications Addressing and Reporting System. http://www.skybrary.aero/index.php/Aircraft_Communications,_Addressing_and_Reporting_System

Lacoste Marc (abril 20 2017). Wikipedia. Flight data recorder. https://en.wikipedia.org/wiki/Flight_recorder#Flight_data_recorder

Scoles, R., "FADEC - Every Jet Engine Should Have One," SAE Technical Paper 861802, 1986, doi:10.4271/861802.

RTCA, Inc (diciembre 13 2011). DO-178C document, http://specs4.ihserc.com/?sess=140641853&prod=SPECS4

Business School. Criterios para elegir tu metodología de desarrollo de software. http://www.obs-edu.com/int/blog-project-management/metodologias-agiles/criterios-para-elegir-tu-metodologia-de-desarrollo-de-software

Camargo Pablo (23 de abril de 2011). Aircraft Articles. Control integral del motor digital (FADEC). http://aircraftarticles.blogspot.com/2011/04/control-integral-del-motor-digital.html?_sm_au_=iqHQ6SNjqKPrDp0M

Turmero Pablo. Monografías. Ingeniería del software embebido. http://www.monografias.com/trabajos105/ingenieria-del-software-embebido-sw/ingenieria-del-software-embebido-sw.shtml

Esterel Technologies (2014). Esterel Technologies. http://www.esterel-technologies.com/products/scade-suite/

Antrim Kate (septiembre 3 2015). Wikipedia. Model-based design. https://en.wikipedia.org/wiki/Model-based_design

Gaius Cornelius (mayo 5 2006). Wikipedia. DO-178B. https://en.wikipedia.org/wiki/DO-178B

Tinoco Gómez, Oscar (2010). Lima, Perú. Revista de la Facultad de Ingeniería Industrial. Criterios de selección de metodologías de desarrollo de software. http://www.redalyc.org/articulo.oa?id=81619984009

BRAUDE (abril 16 2013). Cuarta edición. Metodología en cascada. http://metodologiaencascada.blogspot.com/?_sm_au_=iqHQ6SNjqKPrDp0M

Valdez Huaraca Gerardo, Mendoza Jorge, Torres Alarcón Junior (2013). Pruebas de sistemas y pruebas de aceptación.

https://es.slideshare.net/abnergerardo/pruebas-de-sistemas-y-aceptacion-23663195

Abad Londoño Jorge (abril 06 2005). Ingeniería de software. Tipos de pruebas de software. http://ing-sw.blogspot.com/2005/04/tipos-de-pruebas-de-software.html?_sm_au_=iqHQ6SNjqKPrDp0M

4rsoluciones (noviembre 28 2012). 4rsoluciones. Test de aceptación: el último paso para el aseguramiento de calidad en software. http://www.4rsoluciones.com/blog/test-de-aceptacion-el-ultimo-paso-para-el-aseguramiento-de-calidad-en-software-2/

8. ANEXOS