VNIVERSIDAD NACIONAL AVTÓNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"LAS PRUEBAS DE INTEGRACIÓN EN EL PROCESO UNIFICADO UTILIZANDO MÁQUINAS DE ESTADO"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

OMAR SOTO OCAÑA

DIRECTOR DE TESIS: M. en C. María Guadalupe Elena Ibargüengoitia González

México, D.F. **2011.**

Gracias a DIOS por todas las personas que encontré en este camino…

A mis papás: Ofelia y Armando.

A mis hermanas: Mary y Cata.

A mi novia: Jeanine.

A mi sobrino: Armando.

A todos mis maestros del posgrado representados por mis sinodales, en especial a mi tutora de tesis la M. en C. María Guadalupe Elena Ibargüengoitia González.

A Jackost: Yigal, Esmeralda, Evelia y Rubén.

A todos mis compañeros de la maestría.

A mis padrinos, amigos y compañeros de la Salida.

… todos muchas gracias por ayudarme a lograr conseguir este objetivo.

Índice

Introducción

La Ingeniería de Software abarca diversas disciplinas enfocadas a reducir costos de desarrollo y mejorar la calidad de los productos finales de software. Para asegurar que la calidad sea adecuada, y que los requerimientos del cliente se cumplan, las pruebas tienen como objetivo ayudar a verificar el resultado del software probando cada elemento que se construye, incluyendo elementos internos e intermedios, así como las versiones finales del sistema a ser entregado. Con la calidad se intenta prevenir y remediar defectos además de garantizar un comportamiento adecuado del sistema que se está probando de acuerdo a los requerimientos del mismo.

Las pruebas de software son un elemento para verificar y validar los sistemas de software, por tanto son una actividad esencial en la ingeniería de software. Consisten en demostrar que un programa cumple con las funciones para las que fue creado, ejecutando dicho programa con el objetivo de detectar, y posteriormente corregir, defectos y fallas.

Las opiniones respecto a las pruebas del software han cambiado en una forma más constructiva, se sabe que realizar las pruebas es una tarea que no empieza solamente cuando la fase de programación se ha completado y que no sólo tiene el propósito de detectar errores. Las pruebas del software ahora se ven como una actividad que debe estar presente en todo el proceso de desarrollo del sistema y es una parte importante de la construcción del producto. Son ampliamente utilizadas en la industria para asegurar la calidad, examinar directamente el software en ejecución y proveer una retroalimentación realista de su comportamiento.

Existen diversos tipos de pruebas que se realizan en las diferentes actividades del proceso de desarrollo del software, pero para realizarse se requiere tiempo y presupuesto adicionales, los cuales incrementan el costo total de desarrollo. Por tal motivo las pruebas deben ser planeadas y, de preferencia, estructuradas para que detecten la mayor cantidad de defectos en los sistemas de software y no eleven excesivamente el coste del mismo.

Existen cuatro niveles o granularidades en las pruebas de software, como se mencionará en el primer capítulo del presente trabajo de tesis, el presente trabajo se enfoca en las pruebas de integración. El propósito de las pruebas de integración es verificar que las diferentes unidades, que componen el sistema de software a ser probado, trabajen juntas de manera apropiada.

Por otro lado, una máquina de estados es un modelo del comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entrada actuales sino también del estado actual donde se encuentre el sistema. De tal forma que en

las máquinas de estados se tiene un conjunto de estados que modelan al sistema con un conjunto de entradas y salidas, así con un estado inicial e introduciendo señales de entrada podemos determinar el estado de la máquina en cualquier momento.

Algunas de las propiedades en las máquinas de estado tienen implicaciones prácticas para las pruebas de software, ya que son modelos formales que requieren una especificación explícita para cada conjunto evento-estado. En un modelo en el cual dicho conjunto de evento-estado no está definido se dice que se tiene una especificación incompleta y por tanto se puede decir que es equivalente a un caso de prueba que detecta una falla en el sistema.

En las pruebas de integración se parte de un estado inicial y se requiere que se termine en un estado final o de aceptación equivalente de una máquina de estados, para verificar que el sistema no presenta fallos de integración. En las pruebas de integración se pueden detectar fallas imposibles de encontrar cuando sólo se realizan pruebas de las unidades individuales del sistema de forma aislada.

En los sistemas interactivos, como una aplicación web o un sistema con diferentes opciones en el menú de usuario, es común que en la pantalla principal se presenta un menú de acciones u opciones posibles a realizarse lo que se modela como el estado inicial. Cada opción del menú, se modela como un evento que lleva al siguiente estado del sistema que corresponde a la pantalla que permite efectuar la acción correspondiente a la opción elegida.

El comportamiento final de un sistema interactivo se puede modelar utilizando definiciones de estado, eventos externos, transición y un conjunto de reglas de transición. Cada máquina de estados tiene una expresión regular equivalente, que se puede representar como un autómata o como una tabla. Así, se pueden combinar los métodos formales con las pruebas de integración de un sistema de software para detectar errores y defectos en la integración de clases y paquetes en sistemas de software interactivos.

Al integrar los diferentes componentes de un sistema interactivo, se puede usar la estrategia de integrar cada opción del menú, lo que se puede modelar como una máquina de estados.

Los objetivos del presente trabajo de tesis son dos, el primero es integrar la técnica de máquinas de estado con el Proceso Unificado para realizar las pruebas de integración en los sistemas interactivos. El segundo objetivo es definir una guía para realizar las pruebas de integración en sistemas de software utilizando máquinas de estado en sistemas interactivos.

Para lograr los objetivos del presente trabajo en el Capítulo uno (Pruebas) se presenta un panorama general de las pruebas de software, así como la descripción de los diferentes

niveles de granularidad en las mismas (pruebas unitarias, pruebas de integración, pruebas de sistema y pruebas de aceptación). También se mencionan las actividades que se realizan en el proceso de pruebas como son la planificación, el diseño, la implementación, la realización y el análisis de las pruebas.

En el Capítulo dos (Pruebas de integración en el Proceso Unificado) se presenta una introducción a lo que es el Proceso Unificado, panorama general y ciclo de vida. También se revisan lo que son las pruebas de acuerdo al Proceso Unificado y específicamente las pruebas de integración, el papel de las pruebas en el proceso de software, artefactos y flujo de trabajo.

En el Capítulo tres (Máquinas de estado) se presentan los conceptos generales de las máquinas de estado, autómatas finitos, representación de autómatas finitos como tabla de transición. También se da una introducción a lo que son los elementos básicos para realizar las pruebas de integración con máquinas de estado (estado, transición, evento y salida), el funcionamiento básico para una máquina de estados para sistemas de software y algunas propiedades de los autómatas de estados finitos en las pruebas.

En el Capítulo cuatro (Pruebas con máquinas de estados) se introducen lo que es el modelo de pruebas en el que se revisan lo que son las pruebas utilizando la estrategia de máquinas de estado.

La parte fundamental del presente trabajo se encuentra en el Capítulo cinco (Guía para planear, realizar y analizar las pruebas de integración en el Proceso Unificado utilizando máquinas de estado) en dónde se propone una guía para realizar las pruebas de integración utilizando máquinas de estado siguiendo el Proceso Unificado. En este capítulo se sugieren los siete pasos a seguir para realizar las pruebas de integración utilizando máquinas de estado, pasos que van desde definir el objetivo de la prueba hasta el análisis de los resultados de la misma.

En el Capítulo seis (Aplicación de la guía en el sistema Hemosist), para verificar que la guía propuesta es de utilidad para realizar las pruebas de integración, se realiza la aplicación de la guía a un sistema de software que representa un caso real (Hemosist), sistema de software realizado por los alumnos del Posgrado en Ciencia e Ingeniería de la Computación para el departamento de hemodinámica del Instituto Nacional de Cardiología “Ignacio Chávez”.

Finalmente se presentan las conclusiones del trabajo realizado.

Capítulo 1

Pruebas

El mundo cambia tan rápidamente que se necesita tener una cultura de cambio e innovación. Un área donde se tiene una tremenda innovación es en el campo de las tecnologías de la información. Los sistemas que se desarrollan hoy en día se componen de millones de líneas de código, los procesadores de computadora son mucho mejores y más rápidos que los de hace una década y las redes proveen actualmente un extraordinario ancho de banda.

A pesar de estas innovaciones la necesidad de responder a los riesgos que puedan suceder, por nuestra dependencia a la tecnología, se vuelve trascendental y el tiempo en el que necesitamos responder a éstos disminuye cada día más debido al ritmo de vida que se obtiene con dichas innovaciones. Desde el punto de vista de los sistemas de software, lo anterior significa que necesitamos ser capaces de realizar cambios en la forma de desarrollar software y de probarlo de forma mucho más acelerada. Las pruebas de software deben realizarse de forma rápida y efectiva.

Las pruebas de software son un elemento muy importante para garantizar la calidad de los sistemas de software, por tanto son una actividad esencial en la ingeniería de software. Consiste en demostrar que un programa cumple con las funciones para las que fue creado, ejecutando dicho programa con el objetivo de detectar defectos y fallas.

Las pruebas de software también se pueden definir como el proceso de validación y verificación de que un programa de software cumple con sus objetivos. Se debe tener en cuenta que los objetivos del producto:

1. Cumple con los requerimientos empresariales y técnicos que guiaron a su diseño y a su desarrollo.
2. Funciona como se esperaba.

De acuerdo a la IEEE *las pruebas de software son el proceso de ejecutar un producto para verificar que satisface los requerimientos y poder identificar las diferencias entre el comportamiento real y el comportamiento esperado* (IEEE, 1998). Las pruebas de software permiten identificar cuando el sistema de software no funciona adecuadamente, de acuerdo a los requerimientos. Se considera que una buena prueba es aquella que encuentra la mayor cantidad de errores o defectos, lo cual permite reducir costos.

Las opiniones respecto a las pruebas del software han cambiado en una forma más constructiva, se sabe que realizar las pruebas es una tarea que no empieza solamente cuando la fase de programación se ha completado y que no sólo tiene el propósito de

detectar errores. Las pruebas del software ahora se ven como una actividad que debe estar presente en todo el proceso de desarrollo del sistema y es una parte importante de la construcción del producto. Son ampliamente utilizadas en la industria para asegurar la calidad, examinar directamente el software en ejecución y proveer una retroalimentación realista de su comportamiento.

En términos generales se realizan pruebas para:

- Ahorrar tiempo y dinero. El no realizar las pruebas en el momento adecuado trae consecuencias negativas importantes en el negocio en el que se implementará el sistema. Detectar un error en una fase avanzada puede tener un gran riesgo y ser muy costoso, ya que no solamente se pone en peligro los recursos asignados al proyecto o el proyecto en sí, también peligra todo el ambiente que gira alrededor del sistema. Se pueden perder oportunidades de negocio ya que el tiempo de corregir los defectos finalmente encontrados es considerable y puede suceder que finalmente el propósito para el cual el proyecto fue realizado ya no tenga sentido o sea obsoleto. Además se deben tomar en cuenta los recursos económicos que se asignan a dicho proyecto para realizar las correcciones pertinentes que muchas veces no están consideradas en el presupuesto original.
- Calidad. La calidad es el grado en el que un conjunto de características del software cumple con los requisitos. El asegurar que un sistema realice las funciones para las que fue planeado es uno de los objetivos importantes de las pruebas, para eso se debe tomar en cuenta los requerimientos que se tienen del proyecto y que dichos requerimientos se cumplan en la versión final del sistema. Un elemento crítico para asegurar la calidad en el contexto del proyecto es convertir las necesidades, deseos y expectativas de los interesados en requerimientos definidos en el alcance de las pruebas.
- Detectar fallas lo antes posible para no liberar productos inmaduros. El liberar productos que no están bien probados puede traer consecuencias importantes, como que el cliente no vuelva a tener la confianza que tenía en un principio. Si no se detectan las fallas en una etapa temprana se podrían realizar las correcciones al final, pero esto traería otros errores que no se podrían detectar hasta que el sistema ya se encuentre trabajando en el ambiente final. Lo anterior daría la sensación de tener un producto que no cumple con la calidad requerida y que evidentemente no va a realizar el trabajo para el cual fue creado.
- Reducir costos de soporte y mala imagen que genere con usuarios molestos. Los costos de mantener el software debe conservarse lo más bajo posible, ya que el realizar modificaciones acarrea un costo extra. Además un usuario final, en caso de detectar algún error grave, puede tener la idea de que el sistema no cumple con sus funciones básicas y por lo tanto no le será de utilidad para realizar el trabajo que necesite.

- Mejorar un proceso de desarrollo. El tener la seguridad de que se realizaron las pruebas correspondientes y que con éstas se detectó un buen porcentaje de fallos permite que en proyectos futuros, utilizando las lecciones aprendidas, se apliquen las mismas técnicas que dieron éxito en los primeros, haciendo que los productos realizados sean cada vez de mejor calidad.
- Evitar problemas legales. El no cumplir con lo pactado y no verificar que todos los requerimientos que fueron acordados se cumplan en el proyecto nos puede provocar serios problemas de tipo legal, por lo que es necesario evitar introducir este tipo de problemas a un proyecto donde se espera que logre la satisfacción del cliente y así generar confianza para poder realizar proyectos futuros en forma conjunta.

Los tipos de pruebas se dividen de manera general en **pruebas de verificación** y **pruebas de validación** (J. Rumbaugh, 1995). En las pruebas de verificación se observa si el resultado corresponde a la especificación, es decir, si se está construyendo el sistema de manera correcta. Por otro lado en las pruebas de validación se observa que se está construyendo el sistema correcto, es decir si los requerimientos se están cumpliendo, en otras palabras se asegura que el resultado sea lo que el cliente necesita.

En (Cem Kaner, 2002) se describen cuatro elementos que debe tener una persona en mente al momento de realizar las pruebas:

- Pensamiento técnico: la capacidad de modelar con base en la tecnología y entender las causas y efectos.
- Pensamiento creativo: la capacidad de generar ideas y ver las posibilidades.
- Pensamiento crítico: la capacidad de evaluar ideas y hacer inferencias.
- Pensamiento práctico: la capacidad de poner las ideas en práctica.

Es importante que se tenga en mente las capacidades de la persona que va a probar el software y que además conozca los diferentes niveles de prueba. Se tienen diversos tipos de pruebas que se aplican durante las actividades del proceso de desarrollo de software y cuatro diferentes niveles de pruebas.

Los cuatro niveles de prueba que se utilizan para aplicar las diferentes técnicas existentes son:

- Pruebas de Unidad (Componentes).
- Pruebas de Integración.
- Pruebas de Sistema.
- Pruebas de Aceptación (Validación).

Cabe mencionar que no a todos los sistemas se les puede aplicar todos estos niveles de pruebas, ya que los sistemas son tan complejos y tan diferentes que, en ocasiones, se vuelve innecesario el realizar todos los niveles para sistemas que no lo requieran. Es por esto que, de acuerdo a la visión de la persona encargada de realizar las pruebas, se aplicarán sólo aquellos niveles de pruebas correspondientes al tipo de sistema a probar.

Para ejecutar las pruebas se aplican casos de prueba que son derivados de los requerimientos que se obtienen de los casos de uso cuyo propósito es especificar una forma de probar el sistema mediante un camino que conduzca a la detección de un posible error.

Descripción de los niveles de pruebas

En los diferentes modelos de pruebas para desarrollo de software que se proponen, se tiene el modelo V, figura 1, que se relaciona con cada una de las fases del ciclo de desarrollo del software y con ello es posible preparar la documentación para las pruebas que se realizarán más adelante (plan de pruebas, casos de prueba, procedimientos, etcétera), además de que en este modelo se relaciona cada nivel por el que pasa el proceso de pruebas y su relación con el ciclo de desarrollo del software.

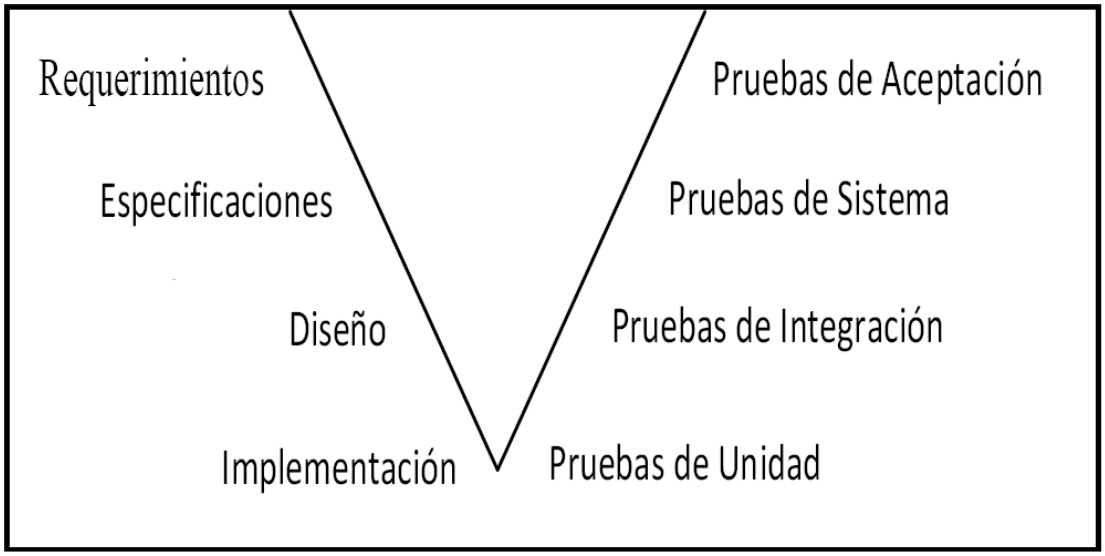

Figura 1. Modelo en V para describir los niveles de prueba

En la figura 1 se muestra cómo en el momento en que el equipo de desarrollo está en la fase de definición de requerimientos con el usuario, el equipo de pruebas define los casos para las pruebas de aceptación con base a los requerimientos que se están definiendo. Una vez que se detallan las especificaciones, el equipo de pruebas prepara los casos de pruebas de sistema. Después en la fase de diseño se preparan las pruebas de integración y finalmente en la implementación se realizan las pruebas unitarias. En cada fase del ciclo de vida del desarrollo de software el equipo de pruebas trabaja para tener listo el ambiente de pruebas que posteriormente se requerirá para la ejecución de las mismas.

A continuación se describe cada uno de los niveles de prueba:

Pruebas unitarias

Las pruebas unitarias verifican el funcionamiento del software de forma aislada de piezas que son comprobables por separado. En función del contexto, estos podrían ser los subprogramas individuales o piezas grandes fabricadas con unidades fuertemente relacionadas.

Durante las pruebas unitarias, la comprobación selectiva de los caminos de ejecución es una tarea esencial. Se deben diseñar casos de prueba para detectar errores debidos a cálculos incorrectos, comparaciones incorrectas o flujos de control inadecuados. Las pruebas del camino básico y de ciclos son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos básicos.

Entre los errores más comunes al realizar los cálculos se encuentran:

- Procedencia aritmética incorrecta o mal realizada.
- Operaciones mezcladas de forma incorrecta.
- Inicialización de variables incorrectas o sin inicializar.
- Falta de precisión.
- Representación simbólica incorrecta de una expresión.

Las pruebas unitarias son las de más bajo nivel. Tradicionalmente, una prueba unitaria consiste en pruebas a procedimientos o subrutinas. En un sistema orientado a objetos se aplican en un nivel más alto, a partir de las clases, los objetos involucran estados encapsulados que pueden afectar el comportamiento y corrección de la unidad.

Tradicionalmente, una prueba unitaria consiste en una prueba estructural también conocida como prueba de caja blanca, la cual requiere conocer el diseño interno de la unidad, y una prueba de especificación también conocida como prueba de caja negra, basada sólo en la especificación del comportamiento externamente visible de la unidad. Normalmente ambas pruebas son necesarias y se complementan entre sí. Los sistemas orientados a objetos pueden utilizar estas pruebas y además requerir otras basadas en estado. Correspondiente al estado encapsulado del objeto y la interacción de las operaciones.

Pruebas de Integración

Después de haber probado todas las unidades de manera independiente mediante las pruebas unitarias, el código se debe integrar en unidades más grandes hasta generar el sistema completo. El propósito de las pruebas de integración es probar que las diferentes unidades trabajen juntas de manera apropiada (Weitzenfeld, 2007), ya que a pesar de que

funcionen correctamente de manera individual no garantiza que al integrarse trabajen correctamente.

Este nivel de pruebas incluye las pruebas por casos de uso, subsistemas, paquetes de servicio, y el sistema completo. No se debe comenzar las pruebas de integración hasta que las pruebas unitarias se hayan terminado.

Las pruebas de integración son una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción de los diversos componentes.

Las pruebas de integración pueden determinar problemas de los siguientes tipos (Binder, 2005):

- Problemas de configuración: Se producen fallas debido a que las versiones de los diferentes elementos pueden resultar incompatibles, por mal control de versiones, o por usar una versión equivocada.
- Funciones faltantes, traslapadas o que tienen conflictos: Una clase puede invocar un método de otra de aún no está implementado o que fue olvidado; también puede suceder que la función invocada no realice lo que se deseaba.
- Uso incorrecto o inconsistente de archivos y bases de datos: Los archivos y bases de datos son elementos importantes al integrar un sistema, pero pueden estar ausentes o tener claves imprevistas o restricción en el número de usuarios concurrentes o formatos diferentes al previsto.
- Violaciones a la integridad de los datos: Al manejar bases de datos, si no se respetan las restricciones de integridad, se producirán errores que quizá no se anticiparon al crear las clases.
- Llamadas a método equivocado, debido a errores de codificación o a liga inesperada al tiempo de ejecución: Como los objetos usan liga dinámica y a veces los nombres de los métodos no dicen su función, es posible invocar métodos equivocados; el polimorfismo lo agrava, ya que no se sabe con exactitud qué objeto será el que interactúe en un momento dado.
- Una unidad cliente envía un mensaje que viola las precondiciones del servidor: por ignorancia o descuido, es posible que una clase solicite un servicio pero que no cumpla las reglas que se necesitan en los parámetros que se envía y eso provoca el rechazo de la clase que debe proporcionar el servicio; por ejemplo esperaba un número positivo y recibe uno negativo.
- Objeto equivocado como destinatario en caso de polimorfismo: se esperaba un objeto y llegó otro y no se sabe qué hacer con él.
- Parámetros erróneos o valores equivocados: los parámetros esperados no corresponden a los enviados; por ejemplo esperaban un número entero y llegó un

número real; otro caso sería que falten parámetros; también puede suceder que el valor no corresponda al rango permitido. Algunos de estos problemas no se notan al compilar debido a la liga dinámica.

- Problema de precisión en parámetros: similar al anterior, pero con tipos parecidos; puede suceder que esperaba un entero de 16 bits y recibe uno de 32 bits o viceversa.
- Fallas causadas por mal manejo de memoria: en ocasiones una clase crea y destruye objetos de otra clase y puede originar problemas si no lo hace correctamente.
- Uso incorrecto de servicios del Sistema Operativo o de máquina virtual: Similares a los anteriores, pero referidos a invocaciones de servicios del sistema operativo o software intermedio, en vez de clases del usuario.
- Conflictos entre componentes: puede ocurrir una falla en una clase cuando otra está corriendo, debido al mal manejo de concurrencia.
- Recursos insuficientes: el destinatario no asigna recursos necesarios, la creación de objetos va disminuyendo los recursos del sistema y puede suceder que se excedan las capacidades asignadas a una aplicación.

Las pruebas de integración atacan directamente a un acoplamiento paulatino de los módulos anteriormente probados en las pruebas unitarias. Las pruebas de integración manejan dos conceptos fundamentales:

Integración Descendente. Ataca directamente a la implementación de los módulos desde el programa principal hacia los módulos inferiores de manera jerárquica. El modo de funcionamiento consiste en que el módulo de control principal maneje toda la información de las pruebas y las distribuya directamente a los módulos que se encuentren inmediatamente subordinados, y a su vez éstos van realizando el mismo procedimiento a sus módulos dependientes. Hay que tener presente que las pruebas se tienen que realizar cada vez que se va incorporando un nuevo módulo.

Integración Ascendente. Como su nombre lo indica, consisten básicamente en tratar el módulo de nivel jerárquico más bajo y de ahí probar hacia arriba. Dado que los módulos son integrados de abajo hacia arriba, el procesamiento requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardo.

También existe lo que se denomina sándwich que es una aproximación combinada que utiliza la integración descendente para los niveles superiores de la estructura del programa junto con la integración ascendente para los niveles subordinados.

Para las pruebas de integración están las pruebas de humo y las pruebas de regresión. El término prueba de humo es el proceso que describe el validar cambios de código antes de que dichos cambios se realicen en la versión final del producto, es un procedimiento de

prueba diseñado para cubrir la mayor parte de la funcionalidad del sistema. Se tiene que las pruebas de humo están diseñadas para confirmar que los cambios en el código funcionan como se espera y no desestabilizan una versión completa. El término "prueba de humo" se originó en la industria del hardware. El término se deriva de esta práctica: después de modificar o reparar un producto o un componente de hardware, el equipo se encendía sin más. Si no salía humo, el componente pasaba la prueba (10Ma).

Por otro lado las pruebas de regresión son una estrategia para probar el software en la cual las pruebas, o por lo menos una parte de ellas, que se han ejecutado anteriormente se vuelven a realizar en la versión modificada, para asegurar que el sistema continúa trabajando correctamente después de añadir la nueva funcionalidad. El objetivo de este tipo de pruebas es garantizar que:

- Los errores encontrados en la ejecución anterior se han corregido.
- Los cambios realizados no han introducido nuevos errores o reintroducido defectos que ya se habían corregido.
- En la práctica, la idea es mostrar que el software que previo a la repetición de las pruebas había demostrado funcionar de forma adecuada lo siga haciendo de la misma forma, cuando se realizan ciertas modificaciones.

La prueba de regresión puede implicar el volver a ejecutar cualquier tipo de prueba realizada anteriormente. Obviamente se debe tomar en cuenta que realizar las pruebas de regresión implica evaluar si el realizarlas aportará una seguridad adecuada y si se tienen los recursos necesarios para realizarlas nuevamente.

La idea general es que teniendo las pruebas unitarias completas, las pruebas de integración se realicen utilizando máquinas de estado. Por lo que el comportamiento final del sistema se modela entonces utilizando definiciones de estado, eventos externos y un conjunto de reglas de transición que soportan el desarrollo de un conjunto de pruebas. Cada máquina de estados tiene una expresión regular equivalente (o un diagrama de secuencia a ser probado).

Las pruebas con máquinas de estado son el objetivo de estudio del presente trabajo, por lo que en el siguiente capítulo se tratará más ampliamente el tema.

Pruebas de Sistema

Este tipo de pruebas se ocupan del comportamiento del sistema completo, es decir, una vez que se han probado los subsistemas del mismo, después de probar todos los casos de uso aislados, se prueba el sistema entero como un todo.

Las pruebas del sistema tienen que ver con el comportamiento de todo el sistema. La mayoría de las fallas funcionales ya debería haber sido identificados durante las pruebas de

integración y las pruebas unitarias. Las pruebas del sistema se consideran apropiadas para comparar el sistema con los requerimientos no funcionales del mismo como son seguridad, velocidad, exactitud y confiabilidad. Las interconexiones externas con otras aplicaciones, utilidades, dispositivos de hardware o con el sistema operativo, también se evalúan en este nivel, sin embargo el tipo de sistema a desarrollar determina qué pruebas son las más adecuadas en este nivel (SWEEBOK, 2004).

Entre las pruebas sistema podemos encontrar:

- Las pruebas de resistencia, que están diseñadas para confrontar los programas con situaciones anormales, en ellas se ejecuta el sistema de tal manera que requiera una frecuencia o un volumen anormal de recursos para verificar que el sistema continúa trabajando. La persona que aplica las pruebas de resistencia tratará de sobrecargar el programa.
- Las pruebas de rendimiento son las pruebas que se realizan para determinar lo rápido que realiza una tarea el sistema en condiciones particulares de trabajo. También puede servir para validar y verificar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos entre otros. Existen distintos tipos de pruebas de rendimiento que ayudan a mejorar las capacidades de la aplicación. Entre las pruebas de rendimiento encontramos:

    1. Las pruebas de carga, las cuales se ejecutan para comprender el comportamiento de una aplicación ante una cantidad de peticiones esperada. Esta carga puede ser el número de usuarios concurrentes utilizando la aplicación web o un número de transacciones durante un tiempo determinado. El resultado de estas pruebas dará el tiempo de respuesta de todas las transacciones críticas.
    2. Las pruebas de estrés son utilizadas normalmente para someter a la aplicación al límite de su funcionamiento, mediante la ejecución de un número de usuarios muy superior al esperado. Estas pruebas tiene como finalidad el determinar la robustez de una aplicación cuando la carga es extrema y ayuda al administrador para determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada. Otro posible objetivo de este tipo de pruebas es determinar el límite real de la aplicación en cuanto a número de usuarios concurrentes, número de transacciones por segundo, etcétera.
    3. Las pruebas de estabilidad se hacen para determinar si la aplicación puede aguantar una carga esperada continua y por un largo tiempo. Generalmente esta prueba se realiza para determinar si hay alguna fuga de memoria en la aplicación.
    4. Las pruebas de picos, tal y como el nombre sugiere, trata de observar el comportamiento del sistema variando el número de usuarios, tanto cuando bajan, como cuando tiene cambios drásticos en su carga.

- Las pruebas de seguridad que verifican que los mecanismos de protección incorporados protegerán al sistema. Un sistema que maneje información que pueda

afectar de forma inapropiada a las personas es un posible objetivo para entradas ilegales que el sistema debe evitar.

- Las pruebas de recuperación son las pruebas que evalúa que tan bien se recupera el sistema luego de bloqueos, fallas del hardware u otros problemas catastróficos. Si la recuperación es automática hay que evaluar la corrección de la inicialización, de los mecanismos de recuperación del estado del sistema, de la recuperación de los datos y del proceso de arranque. Si la recuperación no es automática, hay que evaluar los tiempos medios de reparación para determinar si están dentro de unos límites aceptables. Un sistema tolerante a fallos evita que cese el funcionamiento de todo el sistema cuando se produce un fallo del proceso.
- Las pruebas de portabilidad se refieren al proceso de probar la facilidad con la cual el sistema se puede mover a partir de un ambiente a otro. Esto se mide típicamente en términos de cantidad de esfuerzo permitida. El resultado se expresa en términos de tiempo requerido para mover el software y para terminar actualizaciones de la conversión y de la documentación de datos.

Pruebas de Aceptación

Después que se ha pasado por todas las pruebas de sistema y se han corregido los defectos encontrados, los usuarios y los clientes toman un rol activo en el proceso de pruebas ya que el software se desarrolla precisamente para satisfacer las necesidades del cliente. Las pruebas de aceptación permiten a los usuarios evaluar el software en términos de sus expectativas y metas.

Las pruebas de aceptación verifican el comportamiento del sistema contra los requerimientos del cliente, como éstos se hayan expresado. Los clientes se comprometen a verificar que los requerimientos que pidieron han sido cubiertos o que la organización identificó que estos son necesarios para los objetivos del software. Esta actividad de prueba puede o no involucrar a los desarrolladores del sistema.

Antes de que el software sea liberado es adecuado que el sistema sea revisado por un pequeño grupo que representante a los usuarios que finalmente utilizarán el producto. La mayoría de los desarrolladores de software llevan a cabo lo anterior con un proceso denominado pruebas alfa y pruebas beta para descubrir errores que parezca que sólo el usuario final puede descubrir.

Proceso de las pruebas

El proceso de pruebas involucra consideraciones similares al del proceso desarrollo de software, incluyendo estrategias, actividades y métodos, los cuales deben ser aplicados de manera concurrente con el proceso de desarrollo de software. En particular, las actividades de pruebas tal como lo menciona (I. Jacobson, 2000), consiste en las fases de:

- *Planificar las pruebas* necesarias en cada iteración, incluyendo las pruebas de integración y las pruebas del sistema. Decir que significa planear las pruebas (definir tipos, niveles, estrategias, coberturas, fechas para hacer las pruebas, recursos humanos y económicos, etcétera).
- *Diseñar e implementar las pruebas* creando los casos de prueba que especifican qué probar; los procedimientos de prueba que especifican cómo realizar las pruebas y, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- *Realizar las pruebas y analizar los resultados* de cada prueba sistemáticamente. Los elementos en las que se detecten defectos, se corrigen y son probados de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que los defectos puedan ser modificados, comparando los resultados de las pruebas con los resultados esperados e investigando el porqué de los resultados que posiblemente no coinciden con los resultados esperados. Finalmente se evalúan los resultados y se comparan con los objetivos de las pruebas, se preparan métricas que permiten determinar la compleción de las pruebas y la fiabilidad de las mismas.

Capítulo 2

Pruebas de integración en el Proceso Unificado

El Proceso Unificado es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto (I. Jacobson, 2000).

Panorama general

El Proceso Unificado (UP, por sus siglas en inglés) es el producto de varias décadas de desarrollo y uso práctico. Comenzó desde el Proceso Objectory pasando por el Proceso Objectory de Rational, hasta llegar al Proceso Unificado de Rational (RUP, por sus siglas en inglés). El Proceso Unificado está basado en componentes, es decir que el sistema de software en construcción está formado por componentes de software interconectados a través de interfaces bien definidas, para esto utiliza el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) para preparar todos los esquemas de un sistema de software.

Los tres aspectos definitorios del Proceso Unificado se resumen en tres factores clave (I. Jacobson, 2000):

1. Es dirigido por casos de uso: Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado de valor y representa un requerimiento funcional. Todos los casos de uso juntos constituyen el modelo de casos de uso, el cual describe la funcionalidad total del sistema. Basándose en el modelo de casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo con base en dichos casos de uso. Los desarrolladores revisan cada uno de los sucesivos modelos para que sean adecuados al modelo de casos de uso. Los ingenieros de pruebas verifican la implementación para garantizar que los componentes del modelo de implementación implementan correctamente cada uno de los casos de uso. De este modo, los casos de uso no sólo inician el proceso de desarrollo sino que además le proporcionan un hilo conductor. Por lo anterior los casos de uso no sólo comienzan el proceso de desarrollo, también lo dirigen.
2. Es centrado en la arquitectura: La arquitectura representa la forma, incluye los aspectos estáticos y dinámicos más representativos del sistema. Debe haber interacción entre los casos de uso y la arquitectura, los casos de uso deben encajar con la arquitectura cuando se llevan a cabo y la arquitectura debe permitir el desarrollo de los

casos de uso requeridos, ahora y en el futuro. Tanto la arquitectura como los casos de uso deben evolucionar en paralelo. La arquitectura debe diseñarse para permitir que el sistema evolucione, no sólo en su desarrollo inicial, sino también a lo largo de las futuras generaciones.

3. Es iterativo e incremental: Debido a que el desarrollo de un software puede durar entre varios meses hasta posiblemente un año o más, es práctico dividir el trabajo en partes más pequeños o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento del producto final. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos, al crecimiento del producto. Para una efectividad máxima, las iteraciones deben estar controladas; esto es, deben seleccionarse y ejecutarse de una forma planificada. Es por esto por lo que son miniproyectos. En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes, y verifican que los componentes satisfagan los casos de uso. Si una iteración cumple con sus objetivos el desarrollo continúa con la siguiente iteración. Cuando una iteración no cumple sus objetivos, los desarrolladores deben revisar sus decisiones previas y probar con un nuevo enfoque. Para alcanzar el mayor grado de economía en el desarrollo, un equipo de proyecto intentará seleccionar sólo las iteraciones requeridas para lograr el objetivo del proyecto. Intentará secuenciar las iteraciones en un orden lógico. Un proyecto con éxito se ejecutará de una forma directa, sólo con pequeñas desviaciones del curso que los desarrolladores planificaron inicialmente. Por supuesto, en la medida en que se añaden iteraciones o se altere el orden de las mismas por problemas inesperados, el proceso de desarrollo consumirá más esfuerzo y tiempo. Uno de los objetivos de la reducción del riesgo es minimizar los problemas inesperados.

Estos tres conceptos tienen, cada uno, la misma importancia. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración. La eliminación de una de estas ideas reducirá el valor del Proceso Unificado.

Veamos ahora el Proceso Unificado en su totalidad, su ciclo de vida.

Ciclo de vida del Proceso Unificado

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto para los clientes.

Cada ciclo produce una nueva versión del sistema, y cada versión es un producto preparado para su entrega. Se compone de un cuerpo de código fuente incluido en componentes que puede compilarse y ejecutarse, además de manuales y otros productos

asociados. Sin embargo, el producto terminado no sólo debe ajustarse a las necesidades de los usuarios, sino también a las de todos los interesados, es decir toda la gente que trabajará con el producto. El producto de software debería ser algo más que el código máquina que se ejecuta.

El producto terminado incluye los requerimientos funcionales, especificaciones no funcionales, casos de prueba, el modelo de la arquitectura y el modelo visual. De hecho, incluye todos los elementos que se han mencionado, ya que son esos elementos los que permiten a los interesados especificar, diseñar, implementar, probar y utilizar un sistema. Son dichos elementos los que le permitirán a los usuarios utilizar y modificar el sistema de generación en generación.

Cada ciclo se desarrolla a lo largo del tiempo, dicho tiempo consta de cuatro fases: Inicio, Elaboración, Construcción y Transición (

Figura). Cada fase, a su vez, puede ser dividida en iteraciones con sus incrementos resultantes. Cada fase termina con un hito, el cual se determina por la disponibilidad de un conjunto de artefactos; es decir, de modelos o documentos que indican que se ha alcanzado un estado predefinido.

Los hitos permiten tomar decisiones cruciales antes de que el trabajo pueda continuar con la siguiente fase, así como medir el progreso de dicho trabajo. Al final se obtiene información a partir del seguimiento del tiempo y esfuerzo consumido en cada fase. Estos datos son útiles en la estimación del tiempo y los recursos humanos para otros proyectos, en la asignación de los recursos durante el tiempo que dura el proyecto, y en el control del progreso contrastado con las planificaciones.

En la figura 1 también se muestra en la columna de la izquierda los flujos de trabajo – Requisitos (también conocidos como requerimientos), Análisis, Diseño, Implementación y Prueba. Las curvas son una aproximación de hasta dónde se llevan a cabo los flujos de trabajo en cada fase. Se debe tener siempre en mente que cada fase se divide normalmente en iteraciones o miniproyectos. Una iteración típica pasa por los cinco flujos de trabajo, como se muestra en la figura 1 para una iteración en la fase de elaboración.

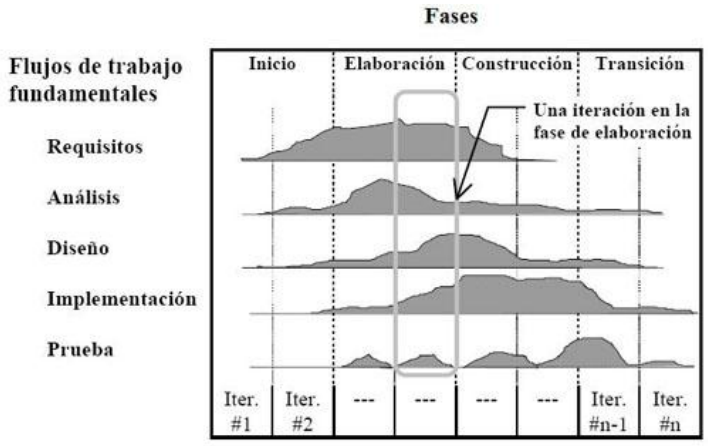

Figura 1 Ciclo de vida del Proceso Unificado

Durante la fase de *Inicio* se desarrolla una descripción del producto final a partir de una buena idea y se presenta el análisis de negocio para el producto. Se realiza una lista de las principales funciones del sistema para sus usuarios, es decir se realiza un modelo de casos de uso simplificado que contenga los casos de uso más críticos. Cuando se tiene, la arquitectura es provisional y consiste en un simple esbozo que muestra los subsistemas más importantes. En esta fase se identifican y priorizan los riesgos más importantes, se planifica en detalle la fase de elaboración, y se estima el proyecto de manera aproximada.

Durante la fase de *Elaboración* se especifica la mayoría de los casos de uso y se diseña la arquitectura del sistema. Se implementan y prueban los elementos más significativos del sistema. Esta fase no solo implica análisis y diseño, se comienzan a programar y a refinar la mayoría de los requerimientos, para que al terminar la fase, estos requerimientos y la arquitectura del sistema hayan sido estabilizados.

Durante la fase de *Construcción* se crea el producto. En esta fase, la línea base de la arquitectura crece hasta convertirse en el sistema completo. La descripción evoluciona hasta convertirse en un producto preparado para ser entregado a la comunidad de usuarios. Se comienza la realización de pruebas de aceptación, la optimización del rendimiento y la creación de documentación para el usuario.

La fase de *Transición* cubre el periodo en el cual el producto se convierte en una versión beta. En la versión beta un número reducido de usuarios con experiencia prueba el producto e informaran de defectos encontrados y proporcionan retroalimentación al equipo de desarrollo. Los desarrolladores corrigen los defectos encontrados y se incorporan las mejoras sugeridas para una versión dirigida a la totalidad de usuarios. La fase de transición conlleva actividades como la fabricación, formación o capacitación del cliente, el proporcionar una línea de ayuda y asistencia, y la corrección de los defectos que se encuentren tras la entrega.

Pruebas en el Proceso Unificado

En el flujo de trabajo de la prueba verificamos el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema a ser entregadas.

Los objetivos de las pruebas, como se mencionó en el capítulo 1, según el Proceso Unificado son (I. Jacobson, 2000)1:

- *Planificar las pruebas.*
- *Diseñar e implementar las pruebas.*
- *Realizar las pruebas y analizar los resultados.*

Los tres puntos anteriores resumen en forma general lo que se busca en una prueba de software y sirven para verificar el resultado de la implementación probando cada construcción así como las versiones finales a ser entregadas.

El papel de la prueba en el ciclo de vida del software

Durante la fase de inicio puede hacerse parte de la planificación inicial de las pruebas cuando se define el ámbito del sistema. Sin embargo, las pruebas se llevan a cabo sobre todo cuando una construcción es sometida a pruebas de integración y de sistema. Esto quiere decir que la realización de pruebas se centra en las fases de elaboración, cuando se prueba la línea base ejecutable de la arquitectura, y de construcción cuando el grueso del sistema está implementado. Durante la fase de transición el centro se desplaza hacia la corrección de defectos durante los primeros usos y a las pruebas de regresión.

Artefactos

- Modelo de pruebas: Describe cómo se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema. El modelo de pruebas es una colección de casos de prueba, procedimientos de prueba y componentes de prueba.
- Caso de prueba: Especifica una forma de probar el sistema, incluyendo la entrada o resultado con la que se ha de probar y las condiciones bajo las que ha de probarse. En la práctica, lo que se prueba puede venir dado por un requisito o colección de requerimientos del sistema cuya implementación justifica una prueba que es posible realizar y que no es demasiado cara de realizar.
- Procedimiento de prueba: Específica cómo realizar uno a varios casos de prueba o partes de éstos. El cómo llevar a cabo un caso de prueba puede ser especificado por un procedimiento de prueba, pero es a menudo útil reutilizar un

[1] Ver el Proceso de pruebas del capítulo 1 el presente trabajo, Pruebas, para conocer los objetivos de las pruebas en el Proceso Unificado.

procedimiento de prueba para varios casos de prueba y reutilizar varios procedimientos de prueba para un caso de prueba.

- Componente de prueba: Automatiza uno o varios procedimientos de prueba o partes de ellos. Los componentes de prueba se utilizan para probar los componentes en el modelo de implementación, proporcionando entradas de prueba, controlando y monitorizando la ejecución de componentes a probar e informando de los resultados de las pruebas.
- Plan de prueba: Describe las estrategias, recursos y planificación de la prueba. La estrategia de prueba incluye la definición del tipo de pruebas a realizar para cada iteración y sus objetivos, el nivel de cobertura de prueba y de código necesario y el porcentaje de pruebas que deberían ejecutarse con un resultado específico.
- Defecto: Es una anomalía del sistema. Un defecto puede ser utilizado para localizar cualquier cosa que los desarrolladores necesitan registrar como síntoma de un problema en el sistema que ellos necesitan controlar y resolver.
- Evaluación de prueba: Es una evaluación de los resultados de los esfuerzos de prueba, tales como la cobertura de la prueba, la cobertura de código y el estado de los defectos.

Flujo de trabajo

El principal objetivo de la prueba es realizar las pruebas como se describe en el modelo de pruebas. Se inicia esta tarea planificando el esfuerzo de prueba en cada iteración, describen entonces los casos de prueba necesarios y los procedimientos de prueba correspondientes para llevar a cabo las pruebas. Si es posible se crean a continuación los componentes de prueba para automatizar algunos de los procedimientos de prueba. Todo esto se hace para cada construcción entregada como resultado del flujo de trabajo de implementación.

1. Planificar la prueba: El desarrollo, realización y evaluación de cada caso de prueba, procedimiento de prueba y componente de prueba lleva algún tiempo y cuesta algún dinero. Sin embargo, ningún sistema puede ser probado completamente; por tanto, deberíamos identificar los casos, procedimientos y componentes de prueba con mayor retorno a la inversión en términos de mejora de calidad. Se debe desarrollar casos y procedimientos de prueba con un solapamiento mínimo para probar los casos de uso más importantes y probar los requerimientos que están asociados a los riesgos más altos. El propósito de la planificación de la prueba es llevar a cabo los esfuerzos de prueba en una iteración llevando a cabo las siguientes tareas:
    - Describir una estrategia de prueba.

- Estimar los requerimientos para el esfuerzo de la prueba, por ejemplo los recursos humanos y software necesarios.
- Planificar el esfuerzo de la prueba.

2. Diseñar prueba: los propósitos de diseñar las pruebas son:
   - Identificar y describir los casos de prueba para cada construcción.
   - Identificar y estructurar los procedimientos de prueba especificando cómo realizar los casos de prueba.

Los casos de prueba de integración se utilizan para verificar que los componentes interaccionan entre sí de la forma apropiada después de haber sido integrados en una construcción. La mayoría de los casos de prueba de integración pueden ser derivados de las realizaciones de casos de uso-diseño, ya que las realizaciones de casos de uso describen cómo interaccionan las clases y los objetos, y por tanto cómo interaccionan los componentes.

Los ingenieros de pruebas deberían crear un conjunto de casos de prueba que hicieran posible alcanzar los objetivos establecidos en el plan de prueba con un esfuerzo mínimo, para realizar esto, los diseñadores de pruebas intentan encontrar un conjunto de casos de prueba con un solapamiento mínimo, cada uno de los cuales prueba un camino o escenario interesante a través de una realización de casos de uso.

Cuando los diseñadores de pruebas crean los casos de prueba de integración consideran como entrada primeramente los diagramas de interacción de las realizaciones de casos de uso. Los diseñadores de pruebas buscan combinaciones de entrada, salida y estado inicial de sistema que den lugar a escenarios interesantes que empleen las clases que participan en los diagramas.

Más tarde, cuando se realiza la prueba de integración correspondiente, tomamos las interacciones actuales de objetos en el sistema. A continuación, comparamos las interacciones actuales con el diagrama de interacciones, las cuales deberían ser iguales; en otro caso se trata de un defecto.

3. Realizar pruebas de integración: En esta actividad se realizan las pruebas de integración necesarias para cada una de las construcciones creadas en una iteración y se recopilan los resultados de las pruebas.

Las pruebas de integración se llevan a cabo en los siguientes pasos:

- Realizar las pruebas de integración relevantes a la construcción realizando los procedimientos de prueba manualmente para cada caso de

prueba o ejecutando cualquier componente de prueba que automatice los procedimientos de prueba.

- Comparar los resultados de las pruebas con los resultados esperados e investigar los resultados de las pruebas que no coinciden con los esperados.
- Informar de los defectos a los responsables de los componentes que se cree contienen los fallos.
- Informar de los defectos a los diseñadores, quienes utilizarán los defectos para evaluar los resultados del esfuerzo de prueba.

4. Realizar prueba de sistema: El propósito es el realizar las pruebas de sistema necesarias en cada iteración y el recopilar los resultados de las pruebas. La prueba de sistema puede empezar cuando las pruebas de integración indican que el sistema satisface los objetivos de calidad de integración fijados en el plan de prueba de la iteración actual.
5. Evaluar prueba: El propósito es el de evaluar los esfuerzos de prueba en una iteración. Se evalúan los resultados de la prueba comparando los resultados obtenidos con los objetivos que se tenían en el plan de pruebas. Se preparan métricas que les permitan determinar el nivel de calidad del software y qué cantidad de pruebas es necesario hacer. En concreto se observan dos métricas:

- Compleción de la prueba, obtenida a partir de la cobertura de los casos de prueba y de la cobertura de los componentes probados. Esta métrica indica el porcentaje de casos de prueba que han sido ejecutados y el porcentaje de código que ha sido probado.
- Fiabilidad, la cual se basa en el análisis de las tendencias en los defectos detectados y en las tendencias de las pruebas que se ejecutan con el resultado esperado.

Las tendencias de los defectos siguen a menudo patrones que se repiten en varios proyectos.

Basándose en el análisis de la tendencia de los defectos se puede sugerir otras acciones, como por ejemplo:

- Realizar pruebas adicionales para localizar más defectos, si la fiabilidad medida sugiere que el sistema no está suficientemente maduro.
- Relajar el criterio para las pruebas, si los objetivos de calidad para la iteración actual se pusieron demasiado altos.
- Aislar las partes del sistema que parecen tener una calidad aceptable y entregarlas como resultado de la iteración actual. Las partes que no

cumplieron los criterios de calidad han de ser revisados y probadas de nuevo.

-

Capítulo 3

Máquinas de estado

Las máquinas de estados han sido ampliamente utilizadas para modelar sistemas en diversas áreas, incluyendo circuitos eléctricos secuenciales, algunos programas para el análisis léxico, para la coincidencia de patrones y hasta en protocolos de comunicación.

Las máquinas de estado son una técnica elemental y efectiva para diseñar y probar un comportamiento de un sistema complejo. Surgieron para describir sistemas reactivos complejos debido a la insatisfacción provocada con notaciones anteriormente empleadas para esa finalidad. (Nogueira de Lucena & Liesenberg, 1996). A pesar de que se tiene un gran número de secuencia de mensajes y de combinaciones en los valores de las variables de instancia, una máquina de estados puede proveer un modelo compacto y predictivo del comportamiento de un sistema.

En este capítulo se dará una introducción a los términos de máquinas de estado que se utilizarán posteriormente.

Autómatas finitos

En esta sección se definirá y estudiará la clase de máquinas teóricas conocidas como autómatas finitos. Aunque su poder es limitado, se encuentra que estas máquinas son capaces de reconocer numerosos patrones de símbolos, los cuales identificamos con la clase de los lenguajes regulares.

Los conceptos que presentan los autómatas finitos y los lenguajes regulares son de interés fundamental para la mayoría de las aplicaciones que requieren técnicas de reconocimiento de patrones. Una de estas aplicaciones es la construcción de compiladores (Brookshear, 1989).

Un autómata finito o máquina de estado finito es un modelo matemático que realiza cómputos en forma automática sobre una entrada para producir una salida (Brookshear, 1989).

Este modelo está conformado por un alfabeto, un conjunto de estados y un conjunto de transiciones entre dichos estados. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

Denotamos formalmente a un autómata finito por el conjunto de cinco elementos (Q, Σ, q0, δ, F) donde:

Q que es un conjunto finito de estados;

Σ es un alfabeto de entrada finito;

q0 ϵ Q es el estado inicial;

δ: Q × Σ → Q es una función de transición que transforma Q × Σ en Q. Esto es, δ(q, a) es un estado que pertenece al conjunto finito de estados (Q) y al que se llega estando en el estado q e introduciendo como entrada un símbolo a que pertenece al alfabeto de entrada finito (Σ);

F es subconjunto de Q que a su vez es un conjunto de estados finales o de aceptación.

Se puede imaginar a un autómata finito como un estado en el que una cinta lectora que avanza sólo hacia delante va leyendo una secuencia de símbolo del alfabeto Σ escritos en la cinta de a una celda a la vez, según la función de transición.

La interpretación de la función de transición δ de un autómata finito determinista es que δ(p, x) = q si y sólo si la máquina puede pasar de un estado p a un estado q al leer el símbolo x (Brookshear, 1989).

En el comienzo del proceso de reconocimiento de una cadena de entrada, el autómata finito se encuentra en el estado inicial y a medida que procesa cada símbolo de la cadena va cambiando de estado de acuerdo a lo determinado por la función de transición. Cuando se ha procesado el último de los símbolos de la cadena de entrada, el autómata se detiene. Si el estado en el que se detuvo es un estado de aceptación o un estado final, entonces la cadena pertenece al lenguaje reconocido por el autómata; en caso contrario se tiene que la cadena dada no pertenece a dicho lenguaje.

Se debe notar que el estado inicial q0 de un Autómata finito siempre será único, en tanto que los estados finales pueden ser más de uno, es decir, el conjunto F puede contener más de un elemento. También puede darse el caso de que un estado final corresponda al mismo estado inicial.

El requisito del determinismo impone restricciones sobre los diagramas de transición que pueden aparecer en los programas para un autómata finito. En particular, cada estado de estos diagramas sólo debe tener un arco que sale para cada símbolo del alfabeto; de lo contrario, una máquina que llega a ese estado se enfrentará a una elección de cuál debe ser el arco a seguir. Además, dicho diagrama deberá estar completamente definido, es decir, debe existir por lo menos un arco para cada símbolo del alfabeto; de lo contrario, una

máquina que llega a ese estado puede enfrentarse a una situación donde no puede aplicarse ninguna transición.

El autómata representado gráficamente en la figura 1 está definido sobre el alfabeto Σ={0,1}, posee dos estados s1 y s2, y sus transiciones son δ(s1,0)=s2, δ(s1,1)=s1, δ(s2,0)=s1 y δ(s2,1)=s2. Su estado inicial es s1 que, en este caso en particular, es también su único estado final. El lenguaje regular que se reconoce con el autómata de la figura 1 puede expresarse mediante la expresión regular (00 | 1 | (01 | 10)(01 | 10)) * .

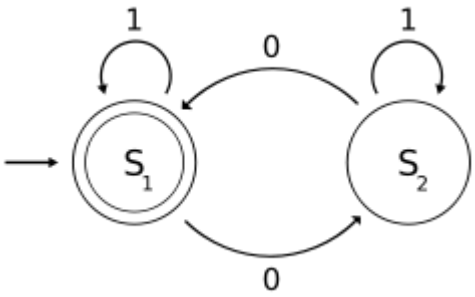

Figura 1. Representación de un diagrama de estados

Los autómatas finitos se pueden representar mediante grafos particulares, también llamados diagramas de estados finitos, de la siguiente manera:

- Los estados se representan como vértices, etiquetados con su nombre en el interior.
- Una transición desde un estado a otro, dependiente de un símbolo del alfabeto, se representa mediante una arista dirigida que une a estos vértices, y que está etiquetada con dicho símbolo.
- El estado inicial se caracteriza por tener una arista que llega a él, proveniente de ningún otro vértice.
- El o los estados finales se representan mediante vértices que están encerrados a su vez por otra circunferencia.

Representación como tabla de transiciones

Otra manera de describir el funcionamiento de un autómata finito es mediante el uso de tablas de transición de estados.

Una tabla de transición de estados es una tabla que muestra que estado (o estados en el caso de un autómata finito no determinista) se moverá la máquina de estados, basándose en el estado actual y otras entradas. Una tabla de estados es esencialmente una tabla en la cual algunas de las entradas son el estado actual, y las salidas incluyen el siguiente estado, junto con otras salidas. Una tabla de estados es una de las muchas maneras de especificar una máquina de estados.

Las tablas de transición de estados son normalmente tablas de dos dimensiones. Hay dos formas comunes para construirlas:

- La dimensión vertical indica los estados actuales, la dimensión horizontal indica eventos o entradas, y las celdas (intersecciones fila/columna) de la tabla contienen el siguiente estado si ocurre un evento (y posiblemente la acción enlazada a esta transición de estados).

Tabla de Transición de Estados				
Entradas Estados	E_1	E_2	...	E_n
q_1	-	A_y /Q_j	. ..	-
q_2	-	-	. ..	A_x /Q_i
...	...	...	. ..	...
q_m	A_z /Q_k	-	. ..	-

(Q: estado, E: evento, A: acción, -: transición ilegal)

- La dimensión vertical indica los estados actuales, la dimensión horizontal indica los siguientes estados, y las intersecciones fila/columna contienen el evento el cual dirigirá al siguiente estado particular. Esta forma de representar los estados como tabla es la que se ocupará en el capítulo 5 para representar los estados.

Tabla de Transición de Estados				
Siguiente estado Estado actual	q_1	q_2	...	q_m
q_1	A_y /E_j	-	. ..	-
q_2	-	-	. ..	A_x/E_i
...	...	...	. ..	...
q_m	-	A_z /E_k	. ..	-

(Q: estado, E: evento, A: acción, -: transición imposible)

Con el autómata finito de la figura 1 se realiza una tabla de transición de estados que se muestra en la siguiente tabla.

Tabla de Transición de Estados		
Entrada / Estado	S_1	S_2
S_1	1	0
S_2	0	1

Representación con máquinas de estados de software

Una máquina de estados puede representar un sistema cuya salida depende de la entrada o evento actual y las entradas o eventos anteriores.

Una máquina de estados es una abstracción de un sistema y está compuesta de cuatro elementos básicos:

Estado

Un estado es una representación de los valores de los atributos y de los enlaces de un objeto, especifica la respuesta del objeto a las entradas siguientes. Un estado corresponde al intervalo entre dos entradas recibidas por el sistema. Las salidas, o eventos, representan puntos temporales, los estados representan intervalos de tiempo.

Al definir estados, se ignoran aquellos atributos que no afecten al comportamiento del sistema/objeto, o que no tengan relevancia en el modelo, y se agrupan en un único estado todas las combinaciones de valores de atributos y de enlaces que tienen una misma respuesta a las entradas.

Evento

Un evento, o entrada al sistema, es lo que causa la transición en los estados. Es la etiqueta de la transición en la gráfica. También se puede ver como la transmisión de información de dirección única en el sistema. Un objeto que envía una entrada a otro objeto puede esperar una respuesta, pero dicha respuesta es a su vez otra entrada completamente distinta, bajo el control del segundo objeto, que puede decidir si la envía o no.

Transición

Una transición lleva a un sistema de un estado a otro. El estado inicial es el estado en el cual inicia el sistema, por lo que no se necesita que se especifique el cómo ingresar a él por medio de una transición, sin embargo el estado actual se refiere al estado activo, es decir al estado al que se llega siguiendo una serie de transiciones con las entradas a través del conjunto de estados que modelan el sistema. Una transición se hace desde un estado actual, activo, a un estado resultante. El estado final es un estado en el cual la máquina se detiene aceptando los eventos que le han ocurrido hasta el momento en el que se llegó a dicho estado.

Salida

La salida de un objeto a una entrada, o evento, puede incluir una acción o un cambio de estado por parte del objeto. Es el resultado que sigue en el estado dependiendo del evento o entrada. Las salidas a un suceso recibido por un objeto pueden variar cuantitativamente, dependiendo de los valores exactos de sus atributos, pero cualitativamente la salida es la misma para todos los valores dentro del mismo estado, y puede ser distinta para valores de distintos estados.

Funcionamiento básico de una máquina de estados para software

Un modelo de máquina de estados válido debe, además seguir las reglas de construcción que a continuación se mencionan, estar diseñado de tal forma que represente el comportamiento real del sistema que se estudia.

El mecanismo de una máquina de estados tiene varios pasos:

- La máquina empieza en el estado inicial.
- La máquina espera por una entrada un intervalo indefinido de tiempo.
- Una entrada se presenta en la máquina.
- Si el evento no está definido en el estado actual, se ignora.
- Si el evento está definido en el estado actual, la transición diseñada se ejecuta: la acción de salida asociada se produce y el estado designado como estado resultado se convierte en el estado actual. El estado actual y el estado resultante pueden ser el mismo.
- El ciclo se repite desde el segundo punto, a menos que el estado resultante sea el estado final.

Del mismo modo el modelo básico de máquina de estado tiene una lista simple de pasos a seguir:

- El proceso por el cual las entradas se generan o se ordenan para ser reconocidos no es un tema del modelo.
- Las entradas se reconocen por separado. Las transiciones se ejecutan una a la vez.
- No se reconocen entradas que no se definieron en el modelo.
- La máquina puede estar en un solo estado a la vez.
- El estado actual no puede cambiar excepto que sea por medio de una transición ya definida.
- El modelo es estático: ningún estado, entrada, salida o transición puede ser movida en el momento de la ejecución de la máquina.
- Los detalles del proceso o la implementación por la cual una salida es computada no es asunto del modelo.

- Intervalos no definitivos se asocian con cualquier aspecto del modelo. El tiempo de ejecución de una transición no consume un tiempo específico.

Con UML se especifica una notación estandarizada para diagramas de estado que puede utilizarse para describir clases, sistemas, subsistemas o incluso procesos de negocio.

El diagrama de estados de UML especifica la secuencia de estados que tienen efecto a causa de cierta secuencia de entradas. Si un objeto se encuentra en cierto estado y se produce una entrada cuyo nombre corresponda al de una de sus transiciones que se definen para dicho estado, entonces el objeto pasa al estado que se encuentra en el extremo de destino de la transición.

Propiedades de los autómatas de estados finitos en las pruebas

Algunas propiedades de los autómatas formales tienen implicaciones prácticas en las pruebas. Por ejemplo, modelos formales requieren una transición especificada explícitamente para cada posible par entrada y estado. Un modelo en el que cualquier par de entrada y estado no están definidos se dice que se tiene una especificación incompleta en el modelo. Casi todos los modelos de estado en ingeniería de software, permiten especificaciones incompletas porque típicamente sólo unos cuantos de todos los posibles pares entradas y estados son de interés. Aunque ésta es una forma conveniente de modelado rápido, se necesita que no se ignoren pares sin especificar para tener todos los casos de pruebas completos.

Un estado Si es alcanzado desde otro estado Sj cuando una secuencia legal de eventos tome la máquina desde Si hasta Sj. Cuando algún estado se dice que es alcanzado sin especificar un estado inicial, el estado inicial se asume. Se debe tomar en cuenta que estados son alcanzables. En la práctica estados inaccesibles surgen por diversas razones:

- Estado muerto. Una vez que la máquina entra a un estado muerto, ésta deja de operar ya que no tiene salida. No hay caminos de salida de un estado muerto, entonces ningún otro estado es alcanzado desde él.
- Ciclo sin fin. Una vez que la máquina de estados entra a un ciclo sin fin, no hay estados que puedan ser alcanzados desde el ciclo. Ese ciclo típicamente previene una transición a un estado final.
- Estado mágico. Un estado mágico no tiene transición de entrada pero provee transiciones a otros estados. Puede describirse como un estado inicial extra, lo que no resulta conveniente.

Un estado muerto o ciclo sin fin son apropiados en algunas raras circunstancias, pero casi siempre son un error. Un estado inaccesible, estado mágico, es siempre un error.

Capítulo 4

Pruebas con máquinas de estados

El garantizar que los sistemas de software continuarán trabajando de manera confiable es la causa por la cual se vuelve necesaria la investigación de las pruebas de software con máquinas de estados, lo anterior se realiza con el objetivo de garantizar el correcto funcionamiento de los sistemas de software y descubrir aspectos importantes de su comportamiento que garanticen una calidad adecuada del producto final.

Emplear pruebas basadas en máquinas de estados no es nuevo, ya que utilizar modelos de estados finitos en el diseño y pruebas de componentes hardware del equipo ha sido largamente establecido y se considera una práctica estándar actual (El-Far, 2001).

Las pruebas utilizando máquinas de estados pueden ser utilizadas para crear un conjunto de casos de prueba de integración para aplicaciones interactivas de sistemas de software en donde se desean probar todos los secuencias de interacciones que tenga el usuario y a través de definir los casos de prueba como secuencias de entradas que se le puedan ingresar a dichos sistemas.

El imaginar a los sistemas de software como máquinas de estados le permite al desarrollador diseñar, documentar y probar rigurosamente un programa en términos de su comportamiento. Por lo anterior un sistema puede dividirse, para su análisis, en un conjunto de estados; en donde cada estado representa una abstracción de dicho sistema en un punto específico de su ejecución, y el comportamiento del sistema cambia de acuerdo a un conjunto de transiciones que parten de un estado inicial y van hacia otros estados. Cada transición corresponde a algún evento externo y que se modela en la máquina de estados para determinar si dicho evento externo tiene o no un efecto en dicho sistema.

El estado que se tiene de una aplicación consiste en el estado colectivo de sus objetos o módulos; por lo que se podrá, una vez que tenemos un estado inicial y tomando en cuenta las entradas externas, determinar cuál es el comportamiento que presentan sus objetos y módulos que influyen en dicho subsistema. Cuando las secuencias de eventos externos se restringen, un funcionamiento correcto de un subsistema debe (Binder, 2000):

1. Aceptar eventos legales. Eventos que se han definido en los casos de prueba y que sólo se deben comprobar.
2. Rechazar eventos que sean ilegales en dicho estado. Eventos que no fueron definidos en los requerimientos del sistema.

3. Producir un estado resultante correcto para aceptar o rechazar eventos. Si el sistema produce un estado de aceptación se puede aprobar el caso de prueba y si no se rechaza.

Modelo de pruebas

El objetivo de modelar un sistema como máquinas de estado es la abstracción del mismo con el fin de simplificar sus funciones. Es por esto que en la integración de algunos sistemas nos enfocamos a la abstracción de un sistema por casos de uso.

Se había definido en el capítulo anterior a un estado como una representación de los valores de los atributos y de los enlaces de un objeto, que especificaba la respuesta del objeto a las entradas.

Para los propósitos del presente trabajo, en sistemas interactivos se define a un estado como una pantalla en el sistema, correspondiente a un caso de uso con un menú de opciones que pueden hacerlo visitar otros casos de uso a través de las pantallas del sistema de otros casos de uso.

En la figura 1 se observa un ejemplo de un sistema interactivo que tiene una pantalla principal con tres opciones en el menú que, al seleccionarlas, redireccionan el sistema a la pantalla de la opción seleccionada.

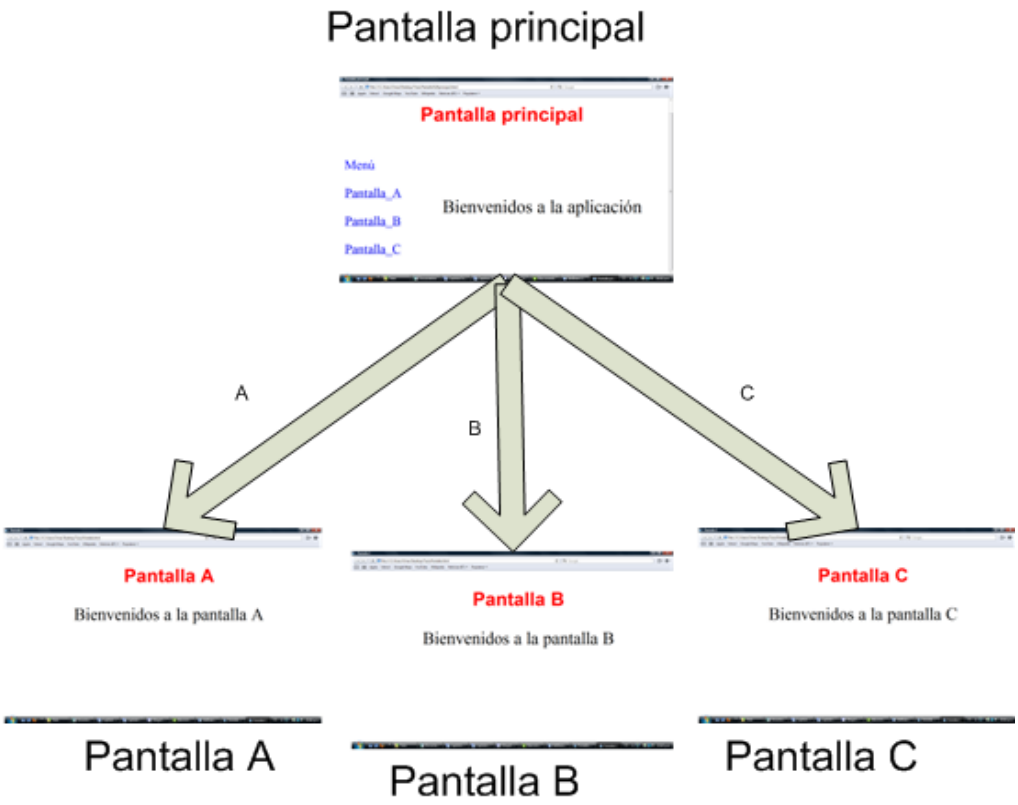

Figura 2. Ejemplo de sistema interactivo que tiene una pantalla principal con tres opciones en el menú.

Utilizando la figura 1 como ejemplo tendremos 4 pantallas o estados, cada una de estas pantallas corresponde a un caso de uso, por lo que la lista de estados quedará:

- *Pantalla principal*
- *Pantalla A*
- *Pantalla B*
- *Pantalla C*

De esta forma, ahora se modela el sistema a través de una tabla con la información de cada uno de los estados de la siguiente forma:

Tabla 1. Identificación de estados del ejemplo de la figura 1.

Estado final / Estado inicial	Pantalla Principal	Pantalla A	Pantalla B	Pantalla C
Pantalla Principal				
Pantalla A				
Pantalla B				
Pantalla C				

Del mismo modo en el capítulo anterior se había definido a un evento, o entrada al sistema, como lo que causa la transición en los estados. Para el presente trabajo se define a un evento como cada una de las opciones del menú en un sistema interactivo que origina el cambio de pantalla o estado.

Con el mismo ejemplo, tenemos que en el menú de la pantalla principal se tiene tres opciones que puede seleccionar el usuario y que representarían a los eventos posibles en el estado *Pantalla Principal*:

- A
- B
- C

Cada una de dichas opciones representa un evento que puede ser seleccionado por los usuarios y que lo pueden llevar, al momento de seleccionarlas, a otro estado.

Por último en el capítulo anterior se definió a una transición como las opciones del menú que llevan a un sistema de un estado a otro dependiendo del evento. Para los propósitos del presente trabajo se tienen que una transición es la selección de una opción del menú, es decir es la acción de seleccionar la opción o el evento que lleva al usuario a cambiar de estado, aunque se podría tener una transición al mismo estado.

Las definiciones anteriores se pueden utilizar para definir casos de prueba y probar la integración de los distintos casos de uso en un sistema interactivo.

La forma en que se propone probar todas las opciones de cada caso de uso, es decir la forma en que se van a definir los casos de prueba para todas las transiciones de cado uno de los estados que componen el sistema, son a partir de la pantalla inicial del sistema que representa el estado inicial. En el estado inicial del sistema se indica hacia qué caso de uso se puede desplazar el usuario. Ya se mencionó que cada una de las opciones del menú en la pantalla principal será un estado al que el usuario pueda llegar a partir del estado inicial y cada una de estas opciones que lleva el sistema de un estado a otro se le conoce como evento, al ser seleccionada se convierte en una transición a partir de dicho estado inicial. El nuevo estado actual tendrá nuevas transiciones.

Todo el sistema se tendrá que recorrer a partir del estado inicial siguiendo transiciones definidas a partir del nuevo estado actual.

El proceso de encontrar la función de transición desde una implementación sigue los siguientes pasos:

1. Identificar las transiciones de estados. En sistemas interactivos se identifican los casos de uso, con sus interfaces correspondientes, y las entradas que le permiten al sistema el moverse hacia otras pantallas del mismo sistema. En el ejemplo de la figura 1 tenemos 4 estados (Pantalla principal, Pantalla A, Pantalla B y Pantalla C) y tres eventos posibles las opciones A, B y C.
2. Identificar el estado inicial (se identifica el estado inicial con una fecha) y el o los estados finales del sistema (los estados finales o de aceptación son identificados con un doble circulo). Básicamente se define la función de transición. En la figura 2 se tiene un ejemplo del autómata del ejemplo de la figura 1.

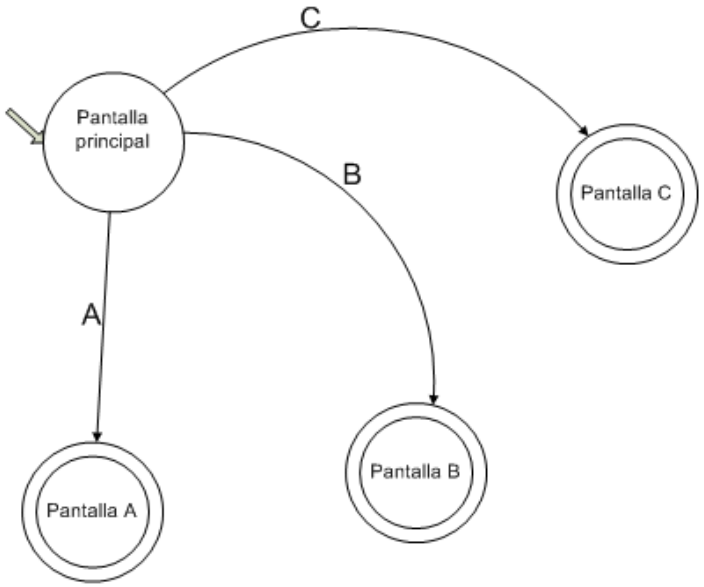

Figura 2 Autómata de sistema interactivo que tiene una pantalla principal con tres opciones en el menú.

Las máquinas de estado son muy útiles, sin embargo cuando se trabaja con sistemas interactivos grandes, modelados con más de 20 estados o casos de uso, se vuelve impráctico trabajar con ellas de forma gráfica. Es por ello que se utiliza la tabla de transición de estados que es un arreglo, o matriz, bidimensional cuyos elementos proporcionan el resumen de una máquina de estados.

Para elaborar la tabla de transición de estados tomamos como base la figura 2 del punto anterior. Para esto se procede a llenar la tabla 2 en la que cada uno de los renglones representa un estado del sistema en un momento dado, las columnas representan el estado al que llega el sistema cuando se introduce la entrada (transición) que se intercepta los renglones (los estados actuales) y las columnas (estado siguiente).

Tabla 2. Ejemplo de tabla de transición para sistema interactivo con tres opciones.

Estado final / Estado inicial	Pantalla Principal	Pantalla A	Pantalla C	Pantalla C
Pantalla Principal	-	A	B	C
Pantalla A	-	-	-	-
Pantalla B	-	-	-	-
Pantalla C	-	-	-	-

Cuando en el sistema no se tiene definida una transición, es decir se está en un estado y no hay un evento que lo lleva a otro estado, en la tabla de transición se deja vacio el espacio.

De esta manera se podrán definir los casos de prueba como eventos para cada estado y comprobar que llevan al estado definido en el autómata. Para realizar lo anterior es conveniente apoyarse en una tabla que resuma algunos puntos como son: el estado inicial del sistema, evento a probar, caso de prueba y por último el estado final esperado después de realizar las pruebas. Se sugiere utilizar la tabla 3.

Tabla 3. Tabla para definir el conjunto de pruebas con máquinas de estado.

[Nombre del sistema a probar]			
Estado inicial	**Evento a probar**	**Caso de prueba**	**Estado final**
[Nombre del estado o pantalla inicial antes de realizar la prueba]	*[Descripción del caso de uso del sistema]*	*[Descripción de lo que el sistema debe realizar]*	*[Nombre del estado o pantalla del sistema al que se llega después de realizar la prueba y acciones sugeridas]*

Si, por ejemplo, se desea probar el caso de uso ingresar a la opción A del sistema se debe establecer el estado inicial (Pantalla principal), el evento a probar (Ingresar a la opción A del menú), describir el caso de prueba (La descripción de lo que el sistema debe realizar)

y por último el estado final del sistema (Pantalla de la opción A). En la tabla 4 se llena la tabla 3 con los datos descritos.

Tabla 4. Tabla de ejemplo para definir el conjunto de pruebas con máquinas de estado del caso de uso ingresar a la opción A del menú.

Sistema de prueba			
Estado inicial	**Evento a probar**	**Caso de prueba**	**Estado final**
Pantalla principal con el menú de opciones	Ingresar a la opción A del menú	El actor selecciona la opción para ingresar a las características de la opción A. La pos condición es que el sistema muestre la página de la opción A.	Pantalla de la opción A

Capítulo 5

Guía para planear, realizar y analizar las pruebas de integración en el Proceso Unificado utilizando máquinas de estado

Las pruebas de integración, como se mencionó en el primer capítulo del presente trabajo, tienen el propósito de probar que las diferentes unidades (que se realizaron en forma independiente) trabajen juntas de manera apropiada. Generalmente se les da poco valor, ya que los desarrolladores son los que regularmente realizan las pruebas de manera informal y éstas se realizan poco antes de entregar el sistema.

Las pruebas de integración se enfocan a la interacción entre unidades, suponiendo que cada una de ellas fue probada a nivel de unidad. En este nivel se mezclan aspectos estructurales que relacionan las maneras de interactuar de las unidades y también otros aspectos funcionales. Además las pruebas de integración pueden detectar problemas como los ya mencionados en el capítulo uno: problemas de configuración, funciones faltantes, uso incorrecto o inconsistente de archivos y bases de datos, violaciones a la integridad de los datos, llamadas a método equivocado, una unidad cliente envía un mensaje que viola las precondiciones del servidor, objeto equivocado como destinatario en caso de polimorfismo, parámetros erróneos o valores equivocados, problema de precisión en parámetros, fallas causadas por mal manejo de memoria, uso incorrecto de servicios del Sistema Operativo o de máquina virtual, conflictos entre componentes y recursos insuficientes. Se debe tomar en cuenta que para realizar las pruebas, las personas encargadas de las mismas, deben tener los cuatro tipos de pensamientos mencionados en el capítulo uno: técnico, creativo, crítico y práctico.

En el primer capítulo también se mencionó lo que son las pruebas por máquinas de estado como técnica para las pruebas de integración. En ellas se tiene un conjunto de entradas y de salidas. Las salidas no sólo dependen de las señales de entrada actuales sino también de las entradas previas. Si tenemos un sistema interactivo (como una aplicación web con diferentes opciones en el menú) se tiene una gran cantidad de interacciones, dependencias y problemas en la navegación de dicho sistema, por lo que su ambiente puede ser completamente diferente a como se encontraba en un principio. Aunque el número de secuencias del menú, las variables y las combinaciones de sus valores es infinito; una máquina de estados puede proporcionar un modelo compacto y fiable de la conducta del sistema (Binder, 2005).

En la primera parte del capítulo 2 se mencionaron las características generales del Proceso Unificado, sus 3 factores clave que lo definen (dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental) y sus fases (Inicio, Elaboración, Construcción y Transición) con sus flujos de trabajo (Requisitos o Requerimientos, Análisis, Diseño, Implementación y Prueba). En dicho capítulo se mencionan los artefactos que intervienen en las pruebas así como el flujo de trabajo para las mismas.

En el capítulo 3 se mencionaron los antecedentes de las máquinas de estado que permiten aplicar dichos conceptos para describir el funcionamiento del software visto como una máquina de estado. El modelar a los sistemas de software como máquinas de estados permite al desarrollador diseñar, documentar y probar rigurosamente un programa en términos de su comportamiento.

Finalmente en el capítulo 4 se presenta el modelo de pruebas con base en máquinas de estado de acuerdo a [Binder] con 4 puntos básicos en el que el comportamiento de los subsistemas es modelado como si fuese una máquina.

Con todos estos elementos como base, se definirá la Guía para pruebas de integración de sistemas interactivos.

Guía de pruebas de integración basada en máquinas de estados

La guía está basada en el comportamiento de los estados de un sistema, definidos por los casos de uso a través de sus pantallas asociadas. Consiste en definir un conjunto de pruebas a realizar en dichos sistemas. Comprenderá actividades desde la planeación de las pruebas según lo propuesto por el Proceso Unificado y complementado con las pruebas de software utilizando máquinas de estado hasta la evaluación de las mismas.

Los pasos para construir el modelo de pruebas de integración propuesto son:

1. Definir el objetivo de la prueba.
2. Entender el sistema a probar.
3. Identificar los estados.
4. Mapear las entradas como eventos que causan las transiciones de estados.
5. Para cada estado de la máquina, definir el conjunto de pruebas y resultados esperados.
6. Aplicar la prueba.
7. Evaluar las pruebas.

1. Definir el objetivo de la prueba.

El principal objetivo de las pruebas es encontrar y corregir errores.

Se inicia esta tarea evaluando cuál es el alcance de la prueba. Para esto se define qué es lo que se quiere probar; ya que ningún sistema puede ser probado completamente, por lo que se deben identificar los casos, procedimientos y componentes de prueba con un mayor retorno de inversión en términos de mejora de calidad. Se deben seleccionar los casos de prueba dentro del número de posibles pruebas donde la probabilidad de encontrar fallas sea alto, considerando las partes del software que son consideradas críticas. También se define qué niveles de integración se pueden probar (clases, componentes, etcétera), para el caso de la guía se espera que la integración sea a nivel de integración entre los casos de uso.

Una vez que se define el alcance de la prueba se debe planificar el esfuerzo y los recursos necesarios para aplicar las pruebas en cada iteración, y se describen entonces los casos de prueba necesarios y los procedimientos de prueba correspondientes. Para planificar los esfuerzos de las pruebas se propone una plantilla que contenga la información necesaria para realizar la prueba, en ella se definen elementos que podrán ayudar a elaborar una adecuada planeación de los esfuerzos y recursos necesarios para realizar ésta tarea.

[Nombre del sistema a probar]	
Prueba de integración	
Objetivos: **Alcance:**	*[Objetivos de realizar las pruebas de integración.]* *[Alcance de la prueba de acuerdo a los objetivos.]*
Recursos requerido	
Sistema a ser probado:	*[Es el sistema o grupo de aplicaciones a ser tomadas en cuenta en la presente prueba.]*
Requerimientos que se toman en cuenta así como los riesgos que conllevan:	*[Son los objetivos específicos que la prueba debe alcanzar. Dichos objetivos deben centrarse en las necesidades del negocio y en los riesgos por lo que se escogen esos requerimientos.]*
Limitaciones del plan de pruebas:	*[Informa de las limitaciones del plan de pruebas, es necesario señalar lo que se pondrá a prueba y lo que no se pondrá a prueba para pronosticar de manera precisa cuales son los recursos que se necesitan utilizar y así poder establecer expectativas adecuadas.]*
Expertos en desarrollo para	*[Alguien del equipo de desarrollo debe ser destinado como un*

planificar las pruebas:	*experto en tecnología para que puede ayudar al equipo de pruebas con la construcción y ejecución de las mismas de forma correcta.]*
Expertos en el negocio para planificar la prueba:	*[No se debe esperar que los encargados de pruebas sean expertos en el negocio, por lo que se pueden apoyar en expertos del negocio para probar parte del sistema. Así que alguien en el "lado del negocio" tiene que ser destinado como un experto en negocios que pueden ayudar al equipo de pruebas con el diseño, ejecución e interpretación de pruebas en el contexto correcto del negocio.]*
Fuente de datos de prueba:	*[Se necesitan datos de prueba lo más cercanos a la realidad, identificar las fuentes y la cantidad de esfuerzo necesario para adquirir los datos para el entorno de prueba.]*
Ambientes de prueba y sus administradores:	*[Después de que el código está escrito y probado por el desarrollador, el código debe realizar la migración a un entorno de prueba independiente para realizar las pruebas correspondientes por parte del equipo de pruebas. Este entorno de prueba independiente debe ser un reflejo del lugar donde funcionará el sistema que se está probando para que los resultados que se arrojen de esta prueba sean confiables.]*

El modelo de casos de uso y los requerimientos adicionales ayudan a decidirse por un tipo adecuado de pruebas y a estimar el esfuerzo necesario para llevar a cabo dichas pruebas.

2. Entender el sistema a probar.

El tener una representación de las funcionalidades del sistema es un requerimiento previo para poder construir los modelos de pruebas. Esta representación no es una tarea trivial ya que la mayoría de los sistemas actuales tienen interfaces complicadas, pero con ayuda de los casos de uso se puede entender el funcionamiento del sistema. Por tal motivo se propone en la guía utilizar una tabla que ayude a entender el sistema a probar. En dicha tabla se sugiere realizar algunas de las actividades para tener una mejor compresión del sistema y poder realizar adecuadamente las pruebas de integración.

[Nombre del sistema a probar]	
Resumen del sistema	
Objetivos:	*[Objetivos del sistema.]*
Alcance:	*[Alcance del sistema.]*

Recursos requerido	
Casos de uso a probar:	*[Se debe determinar los casos de uso a probar con base al objetivo de la prueba.]*
Documentación a la mano:	*[Juntar la documentación pertinente y útil. Ya que las personas que probarán el sistema necesitan entender lo más posible sobre el sistema revisando los requerimientos, los casos de uso, las especificaciones, los documentos diversos del diseño, los manuales de usuario y toda la documentación.]*
Usuarios del sistema:	*[Identificar a los usuarios del sistema, basados en el diagrama de casos de uso.]*
Documentación aplicable para cada entrada:	*[En el caso de sistemas interactivos las entradas son selecciones del menú por parte del usuario, por lo que se debe revisar que las opciones del menú cumplan con los casos de uso del diagrama para cada usuario.]*
Condiciones bajo las que se producen las respuestas a las entradas:	*[Documentar las condiciones bajo las que se producen las respuestas a las entradas. Una respuesta del sistema es una salida o un cambio en sus datos.]*
Comentarios y observaciones:	*[Anotar en el modelo comentarios y observaciones.]*

3. Identificar los estados.

En los sistemas interactivos se trata de modelar todas las opciones del menú que pueden ser seleccionadas por el usuario, para modelarlas como máquina de estados. Para esto, cada caso de uso identificado en el apartado anterior se modelará como un estado y se pondrá en una lista junto con otros estados que se identifiquen en el sistema, esta lista se pondrá en una tabla en la parte de los renglones. Con base en lo planteado en el capítulo 4, Se identifican y generan las listas de:

- Estados (pantallas o casos de uso que se van a probar).
- Entradas (opciones del menú o datos a introducir).

Después de realizar las listas anteriores se tiene que la salida a este punto será el obtener la tabla de estados y entradas para que sirvan de base para el siguiente punto.

4. Mapear las entradas como eventos que causan las transiciones de estados.

A lo largo del tiempo, los objetos se estimulan unos a otros, con las transiciones que se producen por las entradas seleccionadas por los usuarios, dando lugar a una serie de cambios en sus estados. En los sistemas interactivos se tiene que el usuario puede probar

diferentes caminos para lograr la tarea que desea realizar. La manera de probar los diferentes caminos es a través de interactuar con el sistema e introducir diferentes entradas que se eligen del menú de opciones, las cuales son consideradas entradas al sistema.

Según lo expuesto en el capítulo 4 de la presente guía la salida para este punto será el tener la tabla de transición de estados a partir de las listas del punto 3 completa, para que la parte del sistema a probar cumpla con los objetivos propuestos para la prueba.

Para obtener las transiciones de estados se genera una función de transición la cual se obtiene como resultado de los siguientes pasos:

1. Identificar las transiciones de estados. Con base en las listas de estados y de eventos externos identificadas en el punto anterior, se observa y se realiza una nueva lista de los eventos que, al momento de ser introducidos por el usuario, causan transición a algún otro estado.
2.
3. Identificar el estado inicial y el estado final y con esto y los puntos anteriores obtener el autómata finito que simula el trabajo realizado por el sistema que se está probando.
4. Obtener la tabla de transición de estados del autómata finito del sistema. Con base en el autómata encontrado en el punto anterior se realiza la tabla de transición de estados del sistema.

Para probar el caso de uso de ejemplo se muestra la Figura 1 en dónde después de aplicársele la transición A al estado inicial Pantalla principal se llega al estado de aceptación Pantalla A.

Figura 1. Parte del autómata de sistema interactivo que estando en el estado Pantalla principal y al aplicarle el evento externo A lleva al sistema al estado Pantalla A que es de aceptación.

	A
Pantalla Principal	Pantalla A

5. Para cada entrada de la tabla, definir el conjunto de pruebas.

Después de tener la tabla de transición de estados se prosigue a definir el conjunto de datos de prueba para cada una de las posibles entradas en el sistema que se está modelando. Se propone el utilizar una tabla que incluya todas las entradas que causan alguna transición, de acuerdo a la tabla de transición del punto anterior, para cada uno de los casos de uso que se quiere integrar al sistema; en ella se debe presentar el estado inicial del sistema, evento que se va a probar, el caso de prueba que se está probando y por último el estado final del sistema al aplicarle dicha transición. Tal como se muestra en la tabla 1. A partir de la tabla de transición de estados se crea un conjunto de casos de prueba que hagan posible alcanzar los objetivos establecidos en el plan de prueba con un esfuerzo mínimo.

Tabla 1. Tabla de ejemplo para definir el conjunto de pruebas con máquinas de estado.

Sistema de prueba			
Estado inicial	**Evento a probar**	**Caso de prueba**	**Estado final**
Pantalla de bienvenida a los usuarios.	Ingresar al sistema	Debe permitir ingresar al sistema sólo aquellas personas registradas. Pos condición que el usuario ingrese al menú de opciones.	Estado de página principal con el menú de opciones
Pantalla principal con el menú de opciones	Ingresar a la opción A del menú	El actor selecciona la opción para ingresar a las características de la opción A. La poscondición es que el sistema muestre la página de la opción A.	Pantalla de la opción A
Pantalla de la opción A	Salir	El actor selecciona la opción cerrar sesión y el sistema muestra la página principal.	Pantalla de bienvenida a los usuarios.

		La poscondición es que el sistema muestre la página de bienvenida a los usuarios.	

La tabla anterior representa un conjunto de casos de prueba, cada uno de dichos casos de prueba verifica un camino a través de la realización de un caso de uso. Para realizar las pruebas utilizando máquinas de estado se realizan las pruebas de conformidad a cada uno de los casos de uso a probar. El estado que se obtiene después de esta secuencia debe ser el estado esperado.

Lo anterior se inicia con el estado inicial del sistema, el estado inicial consiste en el inicio del sistema, en dónde el usuario puede empezar a trabajar con el sistema interactivo seleccionando alguna opción que se le presente en el menú. Se prosigue a actualizar los estados repetidamente con base en la entrada externa que causa la transición y al estado actual del sistema de acuerdo a la tabla de transición para encontrar el estado. Si se encuentra con un estado que ya ha sido visitado o es el estado final del sistema, entonces no se consideran más transiciones desde ese nodo. Se continúa hasta que se hayan verificado todas las opciones del menú.

La salida a este punto será el tener el plan de pruebas de integración al sistema que se está probando.

6. Aplicar las pruebas.

En esta actividad se realizan las pruebas de integración y se recopilan los resultados de las pruebas con base en el plan de pruebas descrito en la tabla de transición de estados.

Las pruebas de integración se sugiere que se lleven a cabo en los siguientes pasos:

1. Realizar las pruebas identificadas en la tabla de transición de estados que se consideran relevantes para los objetivos de la prueba del primer apartado de la guía realizando los procedimientos de prueba manualmente para cada caso de prueba encontrado.
2. Así con la tabla de transición de estados se va ejecutando el sistema para realizar las pruebas de integración. Después de realizar lo anterior se deben anotar los resultados esperados y finalmente el resultado obtenido.
3. Comparar los resultados de las pruebas con los resultados esperados e investigar los resultados de las pruebas que no coinciden con los esperados.

4. Informar de los defectos a los desarrolladores responsables de los componentes que se cree contiene los fallos para posteriormente puedan ser corregidos.

La salida para este punto será la información necesaria para evaluar la prueba.

7. Evaluar las pruebas.

El propósito de evaluar la prueba es el de valorar si satisface el objetivo de la prueba, para determinar si el software evaluado cumple con la calidad necesaria para posteriores pruebas. La persona responsable de las pruebas evalúa los resultados de la prueba comparando los resultados obtenidos con los objetivos en el plan de prueba.

El análisis de los resultados de las pruebas puede beneficiar a un proyecto de desarrollo de maneras muy diferentes. La primera y más obvia es realizar un seguimiento individual de cada uno de los defectos encontrados para que posteriormente sean corregidos. Con ese seguimiento se proporciona a la persona responsable del desarrollo del sistema la información necesaria para decir el número de defectos descubiertos, el número de defectos corregidos y el número de defectos a la espera de ser corregidos. A partir de los informes de seguimiento, la persona responsable del desarrollo puede tomar decisiones sobre el esfuerzo y la conveniencia de corregir todos los defectos conocidos antes de que el desarrollo se declare terminado.

Basándose en la evaluación de la tendencia de los defectos el encargado de pruebas pueden sugerir otras acciones, como por ejemplo:

- Realizar pruebas adicionales para localizar más defectos, si la fiabilidad medida sugiere que el sistema no está suficientemente maduro.
- Relajar el criterio para las pruebas, si los objetivos de calidad para la iteración actual se pusieron demasiado altos.
- Aislar las partes del sistema que parecen tener una calidad aceptable y entregarlas como resultado de la iteración actual. Las partes que no cumplieron los criterios de calidad han de ser revisados y probados de nuevo.

La persona responsable de las pruebas documenta la compleción de la prueba, su evaluación y sugiere acciones en una descripción de la evaluación de la prueba.

Ajuste de la guía al Proceso Unificado

Después de realizar la guía anterior se tiene que, tal como se reviso en el capítulo 2 del presente trabajo, algunos de las actividades del Proceso Unificado se cumplen en la guía de la siguiente forma:

1. Definir el objetivo de la prueba. En el punto 1 de la guía se cumple el flujo de trabajo de Planificar la prueba, de acuerdo al Proceso Unificado. El propósito de la planificación de la prueba es planificar los esfuerzos de prueba en una iteración llevando a cabo tareas como describir una estrategia de prueba, estimar el esfuerzo requerido para la misma y planificarlo.
2. Entender el sistema a probar. El paso 2 de la guía es parte de las actividades de diseñar la prueba de acuerdo al Proceso Unificado. Los propósitos de Diseñar las pruebas son identificar y describir los casos de prueba e identificar y estructurar los procedimientos de prueba especificando cómo realizar los casos de prueba.
3. Identificar los eventos externos, acciones y mapearlos a estados. Es parte del flujo de trabajo Diseñar la prueba.
4. Mapear las entradas como eventos que causan las transiciones de estados. Es parte del flujo de trabajo de Diseñar la prueba de acuerdo al Proceso Unificado.
5. Para cada estado de la máquina, definir el conjunto de pruebas y resultados esperados. Los puntos 2, 3, 4 y 5 de la guía forman lo que se conoce como Diseñar la prueba en el Proceso Unificado.
6. Aplicar la prueba. El aplicar la prueba tiene el propósito de implementar y de ser posible automatizarla la prueba, tal como se conoce en el Proceso Unificado. Posteriormente en el Proceso Unificado se sugiere realizar la prueba ya que en esta actividad se realizan las pruebas de integración necesarias para cada una de las construcciones creadas en una iteración y se recopilan los resultados esperados.
7. Evaluar las pruebas. El propósito de la evaluación es el de evaluar los esfuerzos de prueba en una iteración. Se evalúan los resultados de la prueba comparando los resultados obtenidos con los objetivos esbozados en el plan de prueba. En la guía esta parte del Proceso unificado se cumple, ya que el flujo de trabajo es el mismo.

Capítulo 6

Aplicación de la guía en el sistema Hemosist

Una vez que se definió la guía para realizar las pruebas de integración en los sistemas interactivos utilizando máquinas de estado, surge la necesidad de probar dicha guía para verificar su adecuado funcionamiento. Por tal razón la guía propuesta en el capítulo anterior ha sido probada en el sistema Hemosist, aplicación realizada por los alumnos del Posgrado en Ciencias e Ingeniería de la Computación de la UNAM para el departamento de Hemodinámica del Instituto Nacional de Cardiología. Cuya definición de requerimientos fueron las siguientes:

Especificación de requerimientos

Definición del problema

El Departamento de Hemodinámica del Instituto Nacional de Cardiología no cuenta con un registro de la información de sus pacientes que los posibilite a realizar análisis estadístico y seguimiento clínico de sus pacientes.

Objetivo del sistema

Desarrollar un sistema de software incluyendo una base de datos para el manejo de los datos de los pacientes del área de hemodinámica del INC y que además quede preparado para incrementar su funcionalidad en ciclos futuros.

Alcance

Hemosist permitirá la captura, modificación y consulta de los datos de los pacientes del departamento de Hemodinámica del Instituto Nacional de Cardiología (INC).

El acceso al sistema estará restringido a médicos registrados, por lo cual se requiere de un módulo de administración que dé de alta a los médicos capturitas.

Definiciones, acrónimos y abreviaturas

Tabla 1. Definiciones, acrónimos y abreviaturas.

Término	Definición
INC	Instituto Nacional de Cardiología
ABCC	Alta, baja, cambio y consulta del término especificado después de las siglas.

ACC	Alta, cambio y consultar del término especificado después de las siglas.
ABC	Alta, baja y cambio del término especificado después de las siglas.

Descripción general

El Departamento de Hemodinámica del INC, necesita un sistema que les permita mantener datos históricos de los pacientes que atienden, incluyendo imágenes y videos de los procedimientos realizados en ellos. Además desean un sistema que les permita la obtención de datos estadísticos que ayuden a las investigaciones que realizan los doctores que trabajan en dicho Departamento.

Para comenzar, el sistema Hermosist sólo permitirá la captura de los datos de generales de los pacientes, la situación médica en la que ingresaron al departamento, los procedimientos y/o intervenciones que le realizaron (incluyendo materiales y medicamentos utilizados) y la situación de los pacientes después de haber egresado.

Debido al presupuesto se ha pedido que el sistema se desarrolle con software libre.

Inicialmente el Departamento no contaba con la infraestructura necesaria para el funcionamiento del sistema, pero se comprometió a tener lista la red y el servidor para la instalación del sistema.

Es importante mencionar que la persona al frente del proyecto está muy interesa en que el sistema sea de fácil captura, lo que él llama “amigable”, ya que tiene la preocupación de que los doctores pierdan interés en utilizar el sistema.

Identificación de los casos de uso

Para la identificación de actores y casos de uso, se tomó como base el documento en Microsoft Office Excel, proporcionado por el doctor a cargo del proyecto. Con la información analizada en el documento antes mencionado se obtuvieron los siguientes casos de uso.

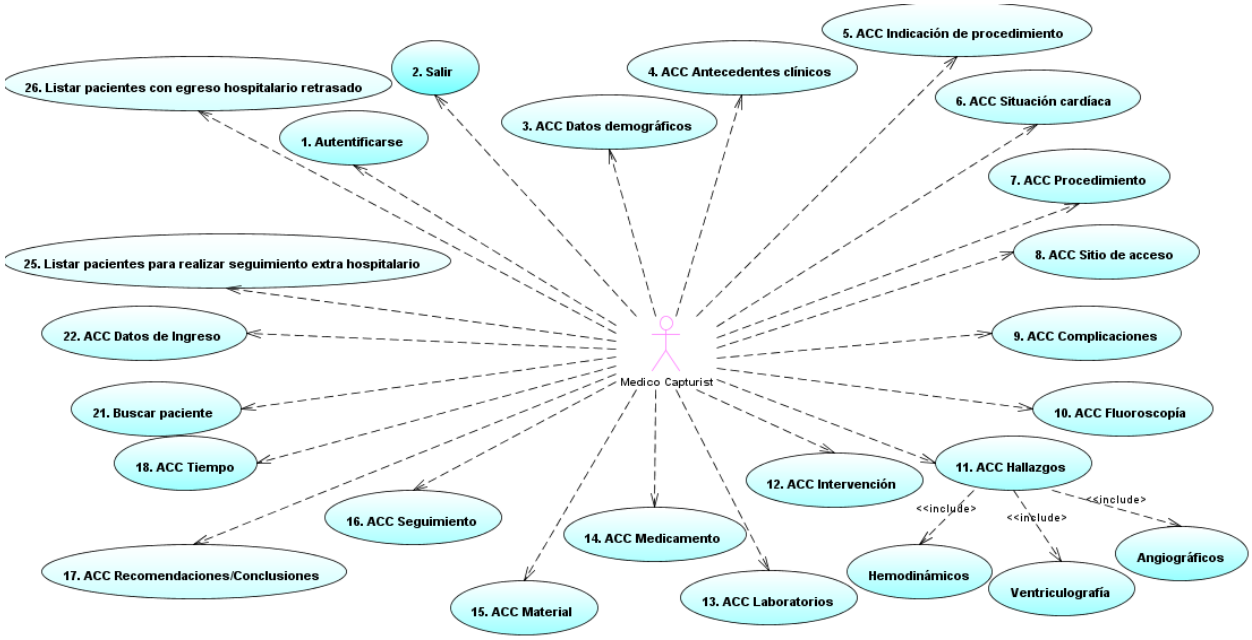

Figura 3. Diagrama general de casos de uso para el actor médico Capturista

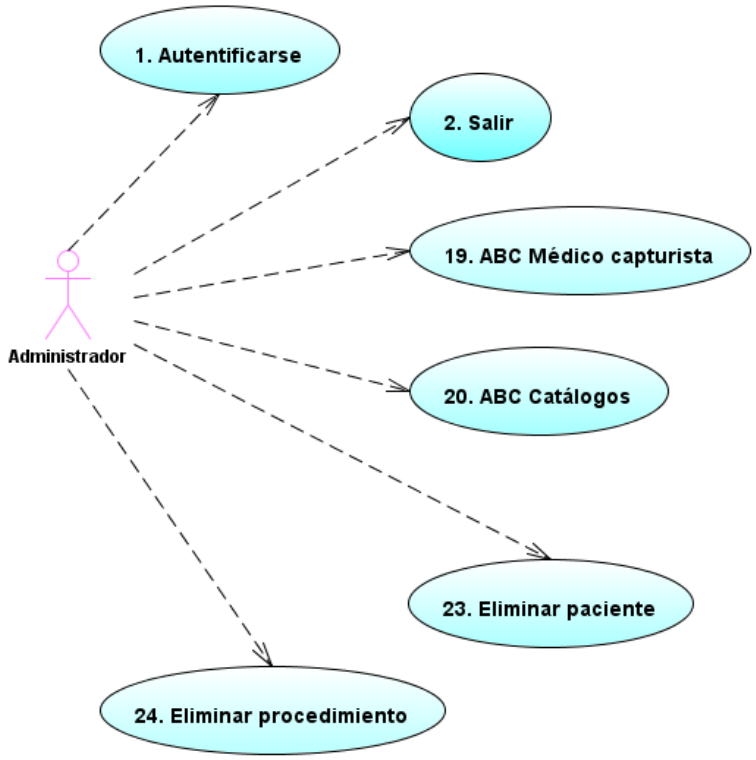

Figura 4. Diagrama general de casos de uso para el actor Administrador

Como se puede observar en el sistema Hemosist se tienen 2 actores, Administrador y Médico Capturista, que originalmente tenían 19 casos de uso para sus funciones, aunque

después se especificaron 7 casos de uso más por petición del cliente para mejorar algunas funcionalidades del sistema dando un total de 26 casos de uso.

Con base en la definición de requerimientos anterior y con la guía propuesta se realizaron las siguientes observaciones: En la pantalla de inicio del sistema Hemosist se les da la bienvenida a los usuarios y se les pide su Nombre (de usuario) y su Contraseña, aquí se tiene el caso de uso número uno (Autenticarse) en el cuál el sistema verifica el usuario y contraseña del usuario y lo redirecciona a cualquiera de los dos tipos de actores que se tiene dentro del sistema para ejecutar los casos de uso que tiene cada uno de ellos. Como se puede observar los dos actores del sistema (Administrador y Médico Capturista) comparten el estado inicial del sistema (Autenticarse) figura 3.

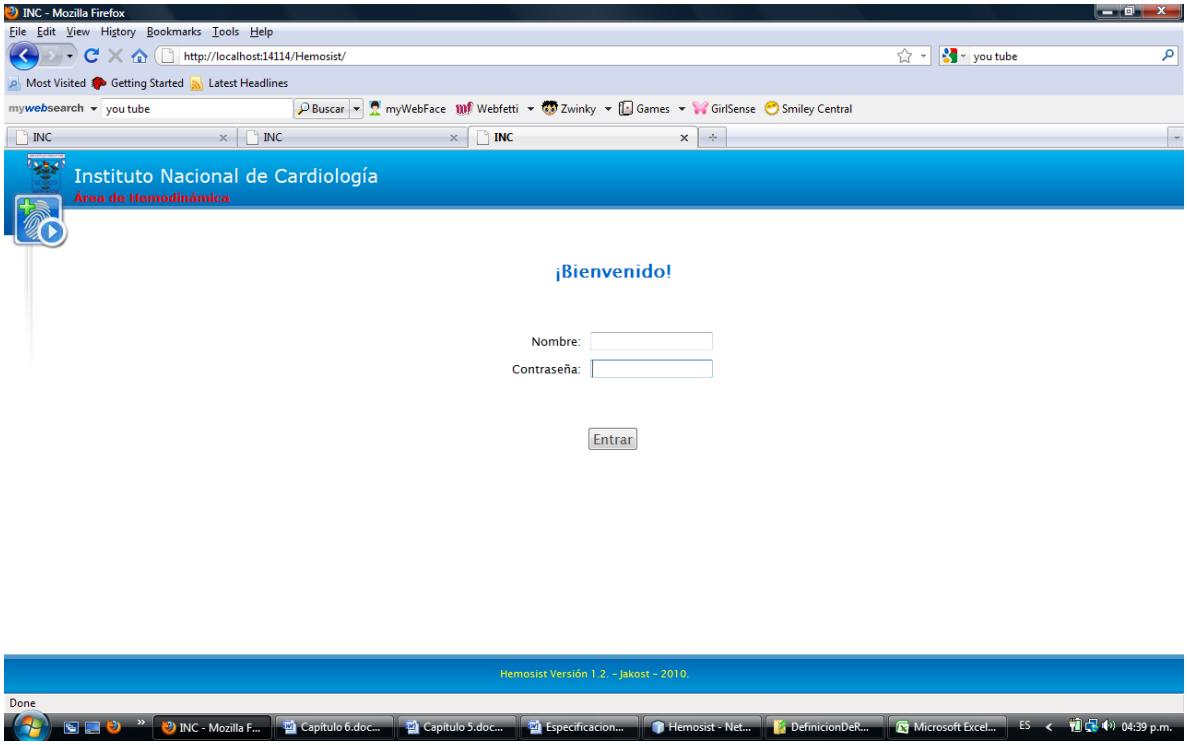

Figura 3. Estado inicial del sistema, caso de uso Autenticarse.

Una vez que el usuario se ha autenticado en el sistema, se presentan dos casos: que el usuario sea el Administrador o que sea el Médico Capturista. En cualquiera de los dos casos en la pantalla principal de cada actor se presente un menú de acciones u opciones posibles a realizarse, cada una de estas opciones generalmente representa un caso de uso. Por ejemplo para el usuario Administrador tenemos 6 casos de uso, aparte del caso de uso Autenticarse que se comparte para ambos actores, Salir, ABCC Medico Capturista, Catálogos, Eliminar paciente y Eliminar procedimiento, figura 4.

Figura 4. Pantalla principal para el usuario Administrador

En la figura 4 se observa que el Administrador además de hacer Altas, Bajas y Cambios de los Médicos Capturistas también puede eliminar pacientes y procedimientos del submenú de pacientes; también puede administrar catálogos, para medicamentos futuros o algún otro catálogo que no se halla considerado como personal médico de reciente ingreso, también tiene el caso de uso Salir ubicado en la esquina inferior derecha etiquetado como "Cerrar sesión". Cada opción del menú se modela como una entrada externa por parte del usuario en donde cada evento de este tipo lleva al siguiente estado del sistema que corresponde a la pantalla y que a su vez permite efectuar la acción correspondiente a la opción seleccionada y llegar a otra funcionalidad del sistema que requiera el usuario, en este caso el del Administrador.

Guía de pruebas de integración basada en máquinas de estados

Tal como se mencionó en el capítulo anterior, la guía está basada en los casos de uso definidos en los requerimientos. Y abarcan las actividades desde la definición del objetivo de la prueba al análisis del resultado de las pruebas según lo propuesto por El Proceso Unificado. Los pasos que se siguieron para construir el modelo de pruebas de integración del sistema Hemosist fueron:

1. Definir el objetivo de la prueba.
2. Entender el sistema a probar.
3. Identificar los estados.

4. Mapear las entradas como eventos que causan las transiciones de estados.
5. Para cada estado de la máquina, definir el conjunto de pruebas y resultados esperados.
6. Aplicar la prueba.
7. Evaluar la prueba.

1. Definir el objetivo de la prueba.

Se mencionó en el capítulo anterior que el principal objetivo de las pruebas es encontrar y corregir errores. En las pruebas de integración el objetivo es verificar que los componentes interaccionan entre sí de la forma apropiada después de haber sido integrados en una construcción.

Para poder documentar el trabajo realizado, y aplicar la guía propuesta, se mostrará sólo la parte del sistema que contiene al actor Administrador del sistema, identificando y probando sus casos de uso (seis para este actor en particular), procedimientos y componentes de prueba que nos darán un retorno de inversión adecuado.

Para la planeación de las pruebas se mencionó en el capítulo anterior se hará uso de la plantilla propuesta:

Hemosist	
Prueba de integración	
Objetivos: **Alcance:**	*Detectar y corregir errores, verificando que los componentes interacciona entre sí de la forma apropiada después de haber sido integrados en una construcción.* *Parte del sistema que contiene los casos de uso para el actor Administrador del sistema.*
Recursos requerido	
Aplicación o sistema a ser probado:	*En este caso en particular probaremos el sistema Hemosist, específicamente la parte del actor Administrador del sistema.*
Requerimientos que se toman en cuenta así como los riesgos que conllevan :	*En el sistema Hemosist se redactaron los documentos DefinicionDeRequerimientosV1.0.docx y EspecificacionDeRequerimientosV1.13.docx para definir los requerimientos del sistema.*
Limitaciones del plan de pruebas:	*En el caso de la guía que se está utilizando se probará únicamente la parte del actor Administrador. Ya que es la parte del sistema que controla los permisos que tienen el actor Médico capturista, además de que al tener sólo seis casos de uso se puede ejemplificar la guía de manera sintetizada.*

Expertos en desarrollo para planificar las pruebas:	*El Ingeniero Administrador de desarrollo.*
Expertos en el negocio para planificar la prueba:	*En el caso del sistema Hemosist esta responsabilidad recayó en el Doctor Yigal Piña Reina, doctor encargado del sistema por parte del Instituto Nacional de Cardiología.*
Fuente de datos de prueba:	*En el caso del sistema Hemosist se utilizaron solamente datos precargados, ya que no se hace uso de archivos o bases de datos externas. El Doctor encargado del sistema por parte del INC verificó los datos que se introducen en toda la aplicación*
Ambientes de prueba y sus administradores:	*Para la aplicación el entorno de pruebas se realizó en el laboratorio de redes y bases de datos en el IIMAS.*

Como se puede apreciar en la plantilla anterior se establecen, en términos generales, las reglas para probar el sistema con base en el alcance de la prueba definido en este mismo punto.

2. Entender el sistema a probar.

Con base en lo visto en el capítulo anterior, se obtiene lo siguiente:

Hemosist	
Resumen del sistema	
Objetivos:	*Desarrollar un sistema de software incluyendo una base de datos para el manejo de los datos de los pacientes del área de hemodinámica del INC y que además quede preparado para incrementar su funcionalidad en ciclos futuros.*
Alcance:	*Hemosist permitirá la captura, modificación y consulta de los datos de los pacientes del departamento de Hemodinámica del Instituto Nacional de Cardiología (INC). El acceso al sistema estará restringido a médicos registrados, por lo cual se requiere de un módulo de administración que dé de alta a los médicos capturitas*
Recursos requerido	
Casos de uso a probar:	*Los cuatro casos de uso a probar son los que se relacionan con el actor Administrador (Autenticar, Salir, ABC Médico Capturista y Eliminar paciente).*
Documentación a la mano:	*Para Hemosist se tienen los siguientes documentos:* • *Plan de la configuración*

	• *Plantilla de solicitud de cambios* • *Plan del proyecto* • *Definición del proyecto* • *Definición de requerimientos* • *Especificación de requerimientos* • *Diseño*
Usuarios del sistema:	*El usuario que se usará para ejemplificar la guía será el Administrador.*
Documentación aplicable para cada entrada:	*En el caso del sistema Hemosist las entradas son selecciones del menú por parte delAdministrador.*
Condiciones bajo las que se producen las respuestas a las entradas:	*Las entradas son opciones del menú, cada una de las entradas al dar de alta algún dato, o modificarse los mismos en la base de datos.*
Comentarios y observaciones:	*Los casos de uso ABC Catálogos y Eliminar procedimiento, que también tienen funciones para el Administrador, no se tomarán en cuenta ya que son muy parecidos a ABC Médico Capturista y Eliminar paciente, respectivamente.*

3. Identificar los estados.

Para la parte de Administrador se tienen 6 casos de uso (Autenticarse, Salir, ABC Médico Capturista, ABC Catálogos, Eliminar paciente y Eliminar procedimiento) que corresponden a los estados de la máquina que modela el sistema en dicha parte. La máquina de estados (en la parte del Administrador) de ABC médico Capturista y ABC Catálogos muestran un comportamiento común para los objetos comunes que se tienen. Se tiene el mismo caso para los casos de uso Eliminar paciente y Eliminar procedimiento. Por lo que sólo se ejemplificará la guía con los casos de uso Autenticarse, Salir, ABC Médico capturista, y Eliminar paciente.

Tabla 2. Identificación de estados.

Identificador	Caso de Uso	Componentes o características que necesitan ser probados
1	Autenticar	Debe permitir ingresar al sistema sólo aquellas personas registradas. Los actores involucrados son el Administrador y el Médico Capturista. La poscondición es que el sistema muestre la página principal para el usuario Administrador o el usuario

		Médico Capturista.
2	Salir	El actor selecciona, Administrador o Médico Capturista, la opción cerrar sesión y el sistema muestra la página principal. La poscondición es que el sistema muestre la página para autenticarse, figura 3.
19	ABC Médico Capturista	El actor Administrador selecciona del menú alguna de las opciones referentes a la administración de Médicos Capturistas. La poscondición es que el sistema muestra la página correspondiente a la selección realizada por el actor, para Alta de médico Capturista, para Cambio y para Baja.
20	ABC Catálogos	El actor Administrador selecciona del menú alguna de las opciones referentes a la administración de catálogo de médicos, Alta Cambio o Baja. La poscondición es que el sistema muestra la página correspondiente a la selección realizada por el actor, Alta Cambio o Baja.
23	Eliminar paciente	El actor Administrador selecciona del paciente a eliminar. Antes ya lo ha buscado con el caso de uso buscar paciente. La poscondición es que el sistema permanece en la pantalla de búsqueda.
24	Eliminar procedimiento	El actor Administrador selecciona el paciente y se muestra sus procedimientos, selecciona alguno para eliminar. La poscondición es que el sistema permanece en la pantalla de búsqueda.

Los casos de uso 19 y 20, ABC Médico Capturista y ABC Catálogos, tienen un comportamiento muy similar; lo mismo ocurre con los casos de uso 23 y 24, Eliminar paciente y Eliminar procedimiento, por lo que la lista de estados quedará:

- Autenticarse.
- Salir.
- ABC Médico Capturista.
- Eliminar paciente.

Los eventos externos y acciones que se identifiquen en cada uno de dichos estados se pondrán en una lista.

Para cada uno de los casos de uso se detectaron los siguientes eventos externos:

Autenticarse:

- Nombre.
- Contraseña.
- Entrar.

Cabe mencionar que el evento externo Nombre también identifica el tipo usuario en el sistema y los privilegios que tiene.

Salir:

- Cerrar sesión.

El caso de uso Salir está presente en toda la aplicación y pueden hacer uso de dicho evento externo en cualquier momento.

ABC Médico Capturista:

- Alta Médico Capturista.
- Nombre del médico.
- Nombre de usuario.
- Contraseña.
- Confirmar Contraseña.
- Cambio Médico Capturista.
- Nombre del médico_A cambiar contraseña.
- Nueva contraseña.
- Confirmar nueva Contraseña.
- Bajas médicos Capturistas.
- Confirmar baja.
- Guardar.
- Cancelar

Eliminar paciente:

- Eliminar paciente.
- Registro.

- Nombre del paciente.
- Apellido paterno.
- Apellido materno.
- Confirmar eliminar paciente.
- Cancelar.

4. Mapear las entradas como eventos que causan las transiciones de estados.

Cada una de las opciones del menú en la pantalla principal del Administrador será un estado al que el usuario podrá llegar a partir del estado inicial, Autenticarse. Al tener un nuevo estado actual será importante el identificar hacia qué estado se puede desplazar el usuario con este nuevo estado actual, interfaz de usuario que se tiene después de haber elegido una opción del menú. Se debe encontrar la forma de recorrer cada una de las interfaces del sistema a través de encontrar la función de transición.

Antes de continuar, se puede notar que el trabajar con el nombre completo de la variable puede llegar a ser muy tedioso y engorroso y no será fácil seguir la secuencia de cada uno de los estados, por lo que a cada uno de los cuatro casos de uso identificados, para realizar las pruebas del actor Administrador, les llamaremos de acuerdo a la codificación de la tabla 3:

Tabla 3. Codificación de casos de uso o estados para el sistema Hemosist para el actor Administrador.

• Autenticarse	• Q1
• Salir	• Q2
• ABC Médico Capturista	• Q3
• Eliminar paciente	• Q4

Del mismo modo a cada uno de los 24 eventos externos encontrados simplemente les llamaremos de acuerdo a la codificación de la tabla 4:

Tabla 4. Codificación de eventos externos para el sistema Hemosist para el actor Administrador.

• Nombre	• E1
• Contraseña.	• E2
• Entrar	• E3
• Cerrar sesión	• E4
• Alta médico Capturista	• E5
• Nombre del médico	• E6
• Nombre de usuario	• E7
• Contraseña	• E8
• Confirmar Contraseña	• E9
• Cambio médico Capturista	• E10
• Nombre del médico_A cambiar contraseña	• E11

• Nueva contraseña	• E12
• Confirmar nueva Contraseña	• E13
• Bajas médicos Capturistas	• E14
• Confirmar baja	• E15
• Guardar	• E16
• Cancelar	• E17
• Eliminar paciente	• E18
• Registro	• E19
• Nombre del paciente	• E20
• Apellido paterno	• E21
• Apellido materno	• E22
• Confirmar eliminar paciente	• E23

El proceso de encontrar la función de transición, para la parte del sistema correspondiente al actor Administrador, siguió los siguientes pasos:

1. Identificar las transiciones de estados. Los casos de uso que se identificaron son los mismos que se presentan en las secciones anteriores y se identificaran con base en la codificación dada en la tabla 3.
    - Q1
    - Q2
    - Q3
    - Q4

    Los puntos de transición son los eventos externos: Entrar, Cerrar sesión, Alta médico Capturista, Cambio médico Capturista, Bajas médico Capturista, Eliminar paciente y Confirmar eliminar paciente. Con base en la codificación de la tabla 4 se tiene la siguiente lista de eventos externos que causan transiciones a otros estados:

• Entrar	• E3
• Cerrar sesión	• E4
• Alta médico Capturista	• E5
• Cambio médico Capturista	• E10
• Bajas médicos Capturistas	• E14
• Eliminar paciente	• E18
• Confirmar eliminar paciente	• E23

2. Se construyó la gráfica de ejecución simbólica. En la figura 5 se presenta el resultado de esta punto en el que se tiene ciertas consideraciones, el estado Q1.1 es un estado en el que entra el sistema una vez que el usuario Administrador se

identificó como tal y espera a realizar los casos de uso Q2, Q3, Q4, Q5 ó Q6. Por otro lado los estados Q3.1, Q3.2 y Q3.3 son subestados (corresponde a Alta Médico Capturista, Cambio Médico Capturista y Baja Médico Capturista respectivamente) que componen el estado ABC Médico Capturista codificacado como Q3. Los eventos que causan cambio de estado son E3, E4, E5, E10, E14, E18 y E23.

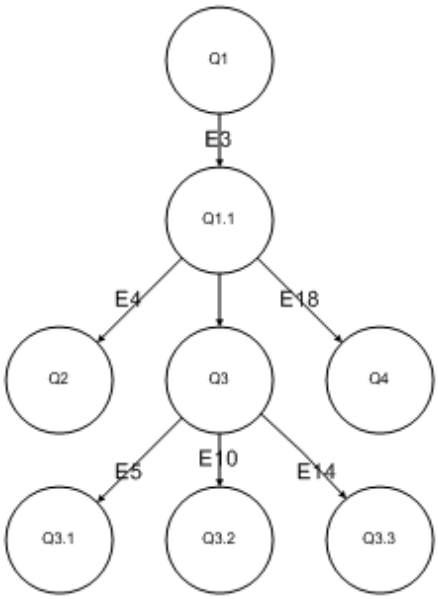

Figura 5. Árbol de ejecución simbólica para la parte del sistema Hemosist correspondiente al Administrador.

3. Se identificó el estado inicial (Q1) y el estado final del sistema (Q2), que al obtener un autómata finito determinista se definió que el estado inicial y final es el estado Q1. Se obtiene el autómata finito con base en la gráfica de ejecución simbólica del punto dos, los estados identificados, las entradas externas que causan transiciones a otros estados y lo visto en el capítulo tres del presente trabajo. En la figura 6 se ve la salida de este punto que es el autómata del sistema.

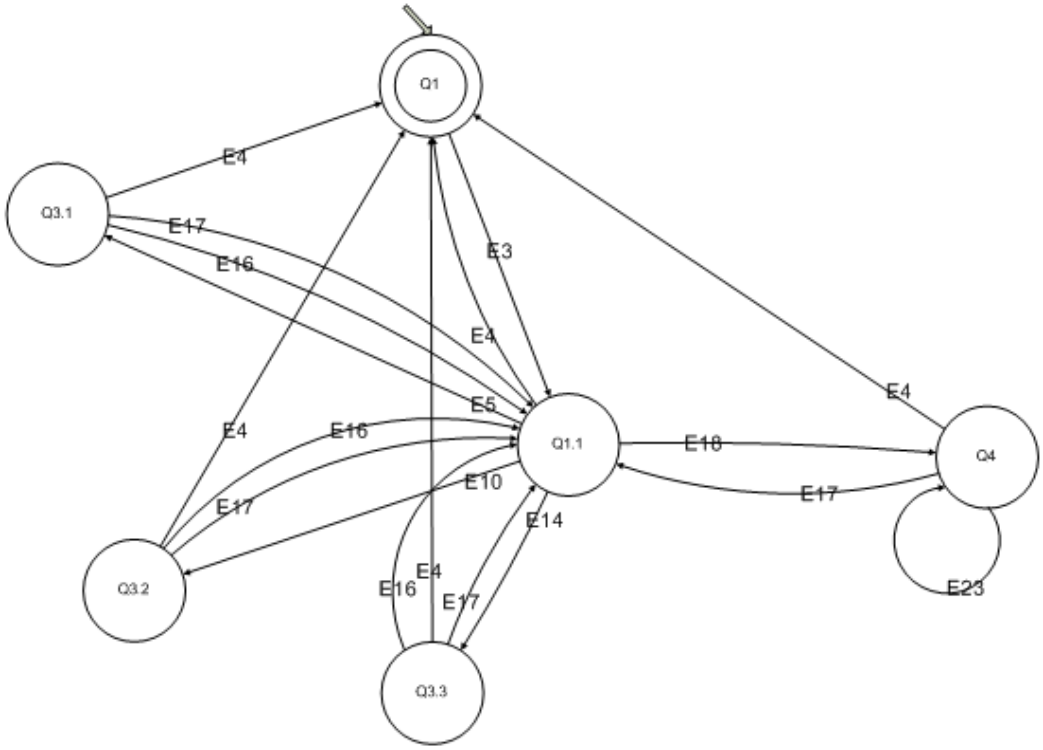

Figura 6. Autómata del sistema Hemosist correspondiente al Administrador.

4. Obtener la tabla del autómata finito del sistema. Con base en la tabla 3 se realizan algunas modificaciones, y se tiene como resultado la tabla 5:

Tabla 5. Identificación de estados y eventos externos del ejemplo del sistema Hemosist para el actor Administrador codificado.

Estado final / Estado inicial	Q1	Q1.1	Q3.1	Q3.2	Q3.3	Q4
Q1						
Q1.1						
Q3.1						
Q3.2						
Q3.3						
Q4						

Ahora lo que se tiene que realizar es completar la tabla 5 poniendo los eventos que ocasionan el cambio de estados para que sea un fiel reflejo del sistema en la parte del Administrador. Para esto, con base en el autómata encontrado en el punto anterior y la tabla 5 se observa que se tiene un estado nuevo definido como Q1.1, que en el sistema Hemosist es una pantalla que muestra el menú de opciones para el actor Administrador una vez que éste se identifico en el sistema como tal. Dicho

estado no es un caso de uso como tal, pero es una parte importante del sistema y que ayuda a tener una visión general de cómo se navega a través del sistema con la parte del actor Administrador.

El caso de uso Q2 (Salir) se eliminó ya que no existe en el sistema una interfaz para dicha acción por lo que una vez que se selecciona la opción o evento externo E4 (Cerrar sesión) el programa nos manda al caso de uso Q1 (Autenticarse).

El caso de uso Q3 (ABC Médico Capturista) se divide en tres casos de uso (Alta Médico Capturista, Baja Médico Capturista y Cambio Médico Capturista), por lo que el estado Q3 desaparece y en su lugar se tienen los estados Q3.1, Q3.2 y Q3.3.

Se tiene que el único estado de aceptación es el estado Q1 (Autenticarse) al que se llega una vez que el usuario presiona la variable E4 (Cerrar sesión) o el tiempo de sesión ha caducado. El estado Q1 también es el estado inicial del sistema, ya que es en él donde se inicia la interfaz de usuario que es la presentación del sistema al usuario.

Las entradas que no están definidas en el sistema, no producen una transición de estado o que no están disponibles en los diferentes casos de uso se marcan en la tabla con un guion (-).

En la tabla 6 se observa la salida final de este punto, la tabla de transición de estados.

Tabla 6. Tabla de transición de estados para el sistema Hemosist.

Estado final / Estado inicial	Q1	Q1.1	Q3.1	Q3.2	Q3.3	Q4
Q1	-	E3	-	-	-	-
Q1.1	E4	-	E5	E10	E14	E18
Q3.1	E4	E16,E17	-	-	-	-
Q3.2	E4	E16,E17	-	-	-	-
Q3.3	E4	E16,E17	-	-	-	-
Q4	E4	E17	-	-	-	E23

Los eventos que llegan al estado Q1.1 a partir del caso de uso ABC Médico Capturista (dividido en Q3.1, Q3.2 y Q3.3) se aprecian que son dos E16 y E17 (Guardar y Cancelar respectivamente) conviene hacer una aclaración, el autómata de la figura 6 es un autómata finito determinista, pero los eventos externos son completamente independientes, E16 y E17, uno del otro.

5. Para cada entrada de la tabla, definir el conjunto de pruebas.

De acuerdo a lo expuesto en el capítulo anterior, para el sistema Hemosist se tiene la tabla 7:

Tabla 7. Tabla para definir el conjunto de pruebas con máquinas de estado para el sistema Hemosist.

Sistema Hemosist			
Estado inicial	**Evento a probar**	**Caso de prueba**	**Estado final**
Q1	Autenticar	Debe permitir ingresar al sistema sólo aquellas personas registradas. Los actores involucrados son el Administrador y el Médico Capturista. La poscondición es que el sistema muestre la página principal para el usuario Administrador o el usuario Médico Capturista.	Q1.1
Q1.1, Q3.1, Q3.2, Q3.3, Q4	Salir	El actor selecciona, Administrador o Médico Capturista, la opción cerrar sesión y el sistema muestra la página principal. La poscondición es que el sistema muestre la página para autenticarse, figura 3.	Q1
Q1.1	Alta Médico Capturista	El actor Administrador selecciona del menú principal de administración de Médicos Capturistas la opción “Alta médico Capturista” (E5). La poscondición es que el sistema muestra la página correspondiente a la selección realizada por el actor, Alta de médico Capturista.	Q3.1
Q1.1	Cambio Médico Capturista	El actor Administrador selecciona del menú principal de administración de Médicos Capturistas la opción “Cambio médico Capturista” (E10).	Q3.2

		La poscondición es que el sistema muestra la página correspondiente a la selección realizada por el actor, Cambio de médico Capturista.	
Q1.1	Baja Médico Capturista	El actor Administrador selecciona del menú principal de administración de Médicos Capturistas la opción "Baja médico Capturista" (E14). La poscondición es que el sistema muestra la página correspondiente a la selección realizada por el actor, Baja de médico Capturista.	Q3.3
Q3.1	Alta Médico Capturista	Una vez que el actor Administrador ingreso los datos del nuevo médico capturista selecciona el botón de "Guardar" (E16). La poscondición es que el sistema notifica que se dio de alta al nuevo médico capturista y el sistema redirecciona al actor Administrador al menú principal del Administrador.	Q1.1
Q3.2	Cambio Médico Capturista	Una vez que el médico capturista está dado de alta en el sistema el actor Administrador cambia la contraseña de dicho médico capturista selecciona el botón de "Guardar" (E16). La poscondición es que el sistema notifica que se cambio la contraseña del médico capturista y el sistema redirecciona al actor Administrador al menú principal del Administrador.	Q1.1

Q3.3	Baja Médico Capturista	Una vez que el médico capturista está dado de alta en el sistema el actor Administrador da de baja los privilegios del usuario médico capturista seleccionando a dicho médico capturista y confirmando que se da de baja con el botón "Guardar" (E16). La poscondición es que el sistema notifica que se dio de baja los privilegios del médico capturista y el sistema redirecciona al actor Administrador al menú principal del Administrador.	Q1.1
Q3.1, Q3.2, Q3.3	Cancelar acción de ABC Médico capturista.	El actor Administrador estando en cualquier interfaz de "ABC Médico capturista" selecciona el botón "Cancelar" (E17). La poscondición es que el sistema lleva al usuario a la interfaz principal del Administrador.	Q1.1
Q4	Eliminar paciente	El actor Administrador selecciona del paciente a eliminar. Antes ya lo ha buscado con el caso de uso buscar paciente. La poscondición es que el sistema permanece en la pantalla de búsqueda.	Q4
Q4	Cancelar el eliminar paciente.	El actor Administrador estando en la interfaz de usuario de "Eliminar paciente" selecciona el botón "Cancelar" (E17). La poscondición es que el sistema lleva al usuario a la interfaz principal del	Q1.1

		Administrador.	

En el capítulo anterior se mencionó que se deben realizar las pruebas de conformidad utilizando máquinas de estado y para el sistema Hemosist se definió lo siguiente en cada uno de los casos de uso:

Pruebas de conformidad: Para cada uno de los 4 casos de uso que fueron identificados en la tabla de transición de estados se trazara un camino para cada una de las secuencias legales de eventos:

- Autenticarse: Se ingresará al sistema con un usuario de tipo Administrador y se presentará la interfaz de menú del usuario Administrador.
- Salir: En cada uno de los casos de uso identificados (Q1.1, Q3.1, Q3.2, Q3.3, Q4, Q5 y Q6) se saldrá del sistema hasta llegar nuevamente al estado final Q1 (Autenticarse, que también es el estado inicial del sistema).
- ABC Médico Capturista: Se dará de alta un médico Capturista de los que se tenga disponibles en el menú de médicos, se modificara y finalmente se dará de baja el mismo médico Capturista. Una vez que se llegue al estado Q1.1 o Q1 habiendo realizado las operaciones designadas se tiene que a partir de dichos estados se llega al estado de aceptación de acuerdo a los primeros casos de prueba (Autenticarse y Salir).
- Eliminar paciente: De la base de datos se eliminará a algún paciente, que los usuarios Médicos Capturistas hayan dado de alta previamente. Una vez que se regrese al estado Q1.1 o Q1 habiendo realizado las operaciones designadas se tiene que a partir de dichos estados se llega al estado de aceptación de acuerdo a los primeros casos de prueba (Autenticarse y Salir).

El estado que se obtiene después de esta secuencia debe ser el estado esperado, es decir el sistema debe permitir realizar todas las actividades que se requieran y posteriormente llegar al estado de aceptación. El estado inicial en el caso del sistema Hemosist corresponde también al estado de aceptación del sistema. Con base en la tabla de transición de estados se debe encontrar el estado resultante para cada estado cuando se elija una opción en el menú. Si se encuentra con un estado que ya ha sido visitado o es el estado final, entonces no se consideran más transiciones desde ese nodo. Se continúa hasta que se hayan verificado todas las opciones del menú.

6. Aplicar las pruebas.

En el sistema Hemosist se realizaron las pruebas de integración bajo los pasos del punto anterior.

Las pruebas de integración se sugiere que se lleven a cabo en los siguientes pasos:

1. Realizar las pruebas identificadas en la tabla del apartado anterior que se consideran relevantes para los objetivos de la prueba del primer apartado de la guía (Probar la parte del sistema de Administrador) realizando los procedimientos de prueba manualmente para cada caso de prueba encontrado.
   - Autenticarse. Se ingresará al sistema con un usuario de tipo Administrador y se presentará la interfaz de menú del usuario Administrador.
     Como objetivo de la prueba revisamos que la funcionalidad implementada corresponda a todo el caso de uso, la parte del autómata que se identificó para este caso de uso (Autenticarse) se muestra en la Figura 7.

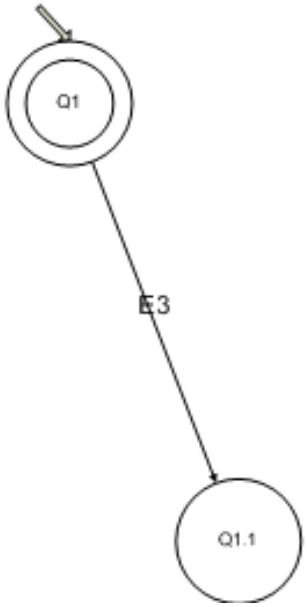

Figura 7. Parte del autómata que se encontró en el punto cinco de la guía y que corresponde al caso de uso Autenticarse.

La parte de la tabla que identifica esta parte del sistema se muestra a continuación:

Tabla 8. Tabla de transición de estados para el sistema Hemosist, Autenticarse.

Estado final / Estado inicial	Q1	Q1.1
Q1	-	E3

La secuencia comienza en el estado Q1 (Autenticarse) que corresponde al estado inicial del sistema, la parte inicial del mismo. Se tienen que la variable E3 (Entrar), si se introducen los datos correctos, logra que el sistema cambie al estado Q1.1. No se tiene una transición que, estando en el estado Q1, la lleve al mismo estado.

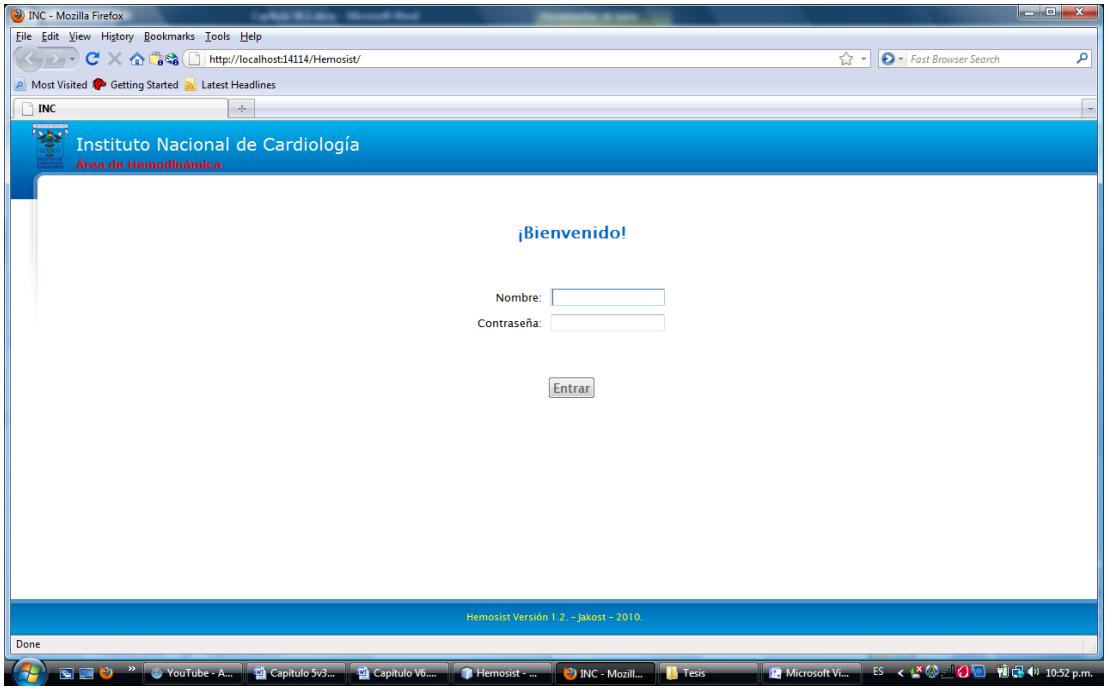

Figura 8. Pantalla inicial del sistema Hemosist.

Si el usuario no introduce su nombre de usuario o contraseña de forma correcta, sistema no permite un cambio de estado Figura 9.

En caso de que el usuario Administrador esté dado de alta como tal en la base de datos del sistema, de su nombre y contraseña correctos se muestra el menú de opciones para el usuario Administrador Figura 10.

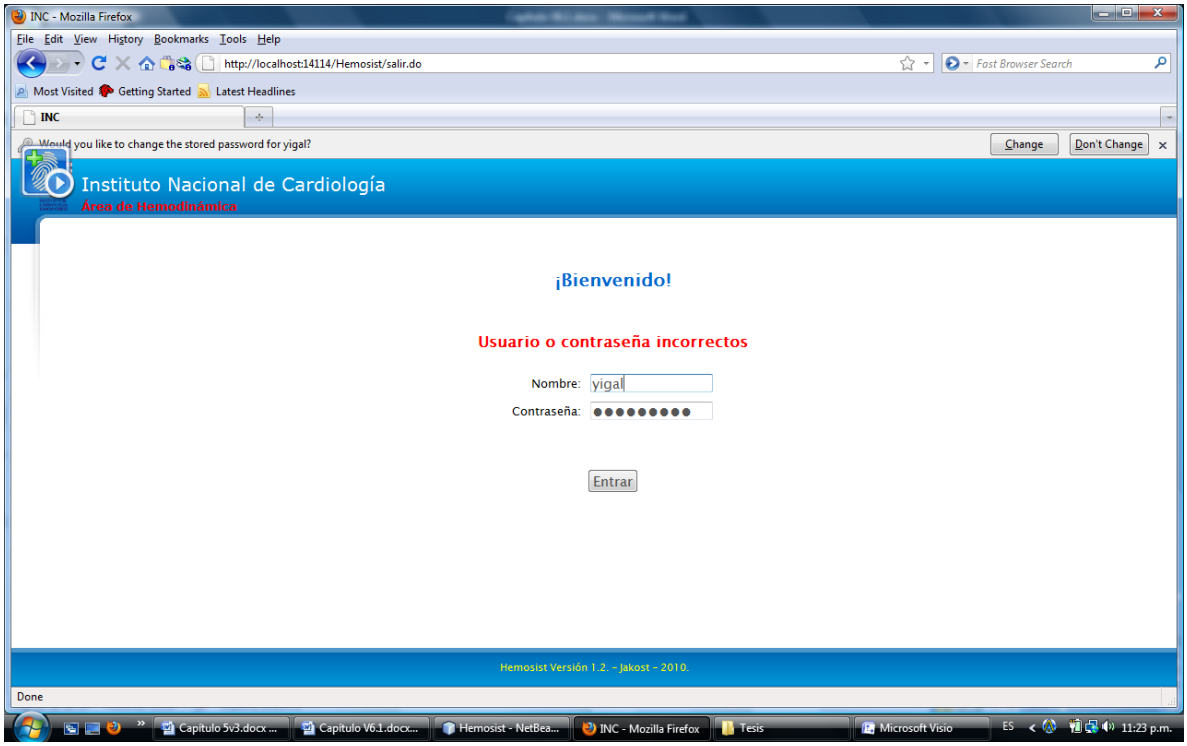

Figura 9. Pantalla inicial del sistema Hemosist que no cambia de estado en caso que el nombre de usuario o la contraseña sean incorrectos.

Figura 10. Pantalla que muestra el menú de opciones para el usuario Administrador, una vez que éste se Autentico en el sistema.

- Salir: En cada uno de los casos de uso identificados (Q1.1, Q3.1, Q3.2, Q3.3, Q4, Q5 y Q6) se saldrá del sistema hasta llegar nuevamente al estado final Q1 (Autenticarse, que también es el estado inicial del sistema).
  Como objetivo de la prueba revisamos que la funcionalidad implementada corresponda a todo el caso de uso, la parte del autómata que se identifico para este caso de uso (Salir) se muestra en la Figura 11.

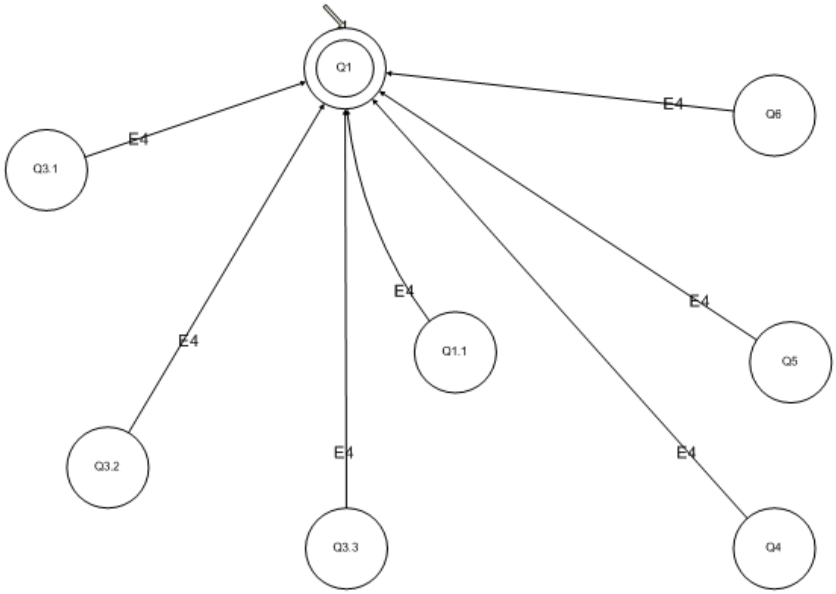

Figura 11. Parte del autómata que prueba el caso de uso Salir en caso de que se le aplica la entrada externa E4 (Cerrar sesión).

El autómata y la parte de la tabla que identifica esta parte del sistema aclaran que estando en cualquier estado del sistema al aplicarle la entrada E4 (Cerrar sesión) se llega al estado de aceptación Q1 (Autenticarse), se muestra a continuación la parte de la tabla que identifica esta parte del sistema:

Tabla 9. Tabla de transición de estados para el sistema Hemosist, Salir.

Estado final / Estado inicial	Q1
Q1	-
Q1.1	E4
Q3.1	E4
Q3.2	E4
Q3.3	E4
Q4	E4

Estando por ejemplo en el recorrido del autómata se tiene que estando en el estado Q1.1 y presionando la variable E4 (Cerrar sesión, ubicada en todo el

sistema en la parte inferior derecha) se llega al estado de aceptación Q1 (Autenticarse) Figura 12.

Figura 12. El evento o entrada externa E4 (Cerrar sesión) se encuentra en todos los estados del sistema en la parte inferior derecha.

De ahora en adelante se probarán todos los estados o casos de uso sabiendo, por la prueba del caso de uso Salir del sistema, que todos los estados pueden llegar al estado de aceptación Q1 (Autenticarse), por la variable E4 (Cerrar sesión que se encuentra en todos los estados. Si se encuentra con un estado que ya ha sido visitado o es el estado final, entonces no se consideran más transiciones desde ese nodo.

- ABC Médico Capturista: Se dará de alta un médico capturista de los que se tenga disponibles en el menú de médicos, se modificará y finalmente se dará de baja el mismo médico capturista. Una vez que se llegue al estado Q1.1 o Q1 habiendo realizado las operaciones designadas se tiene que a partir de dichos estados se llega al estado de aceptación de acuerdo a los primeros casos de prueba (Autenticarse y Salir).

  Para realizar las pruebas, en este caso de uso en particular, debemos recordar que el estado se subdividió en tres estados Q3.1, Q3.2 y Q3.3.

  La parte del autómata que verifica el caso de uso Q3.1 se muestra en la Figura 13.

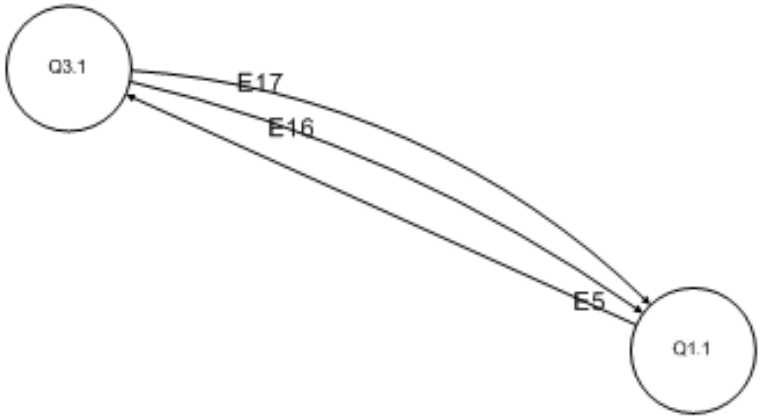

Figura 13. Parte del autómata que prueba el caso de uso Q3.1 Alta Médico Capturista.

Llegamos al estado Q3.31 al introducir la variable E5 (Alta médico Capturista). La parte de la tabla, correspondiente al autómata de la Figura 13 se muestra a continuación:

Tabla 10. Tabla de transición de estados para el sistema Hemosist, Alta Médico Capturista.

Estado final / Estado inicial	Q1.1	Q3.1
Q1.1	-	E5
Q3.1	E16,E17	-

Las variables E16 y E17 (Guardar y Cancelar respectivamente) hacen una transición al estado Q1.1, en caso de que los datos introducidos sean válidos, y si no se muestra el mensaje de error en el mismo estado figuras 15 y 16.

Figura 14. Caso de uso Q3.1 Alta Médico Capturista.

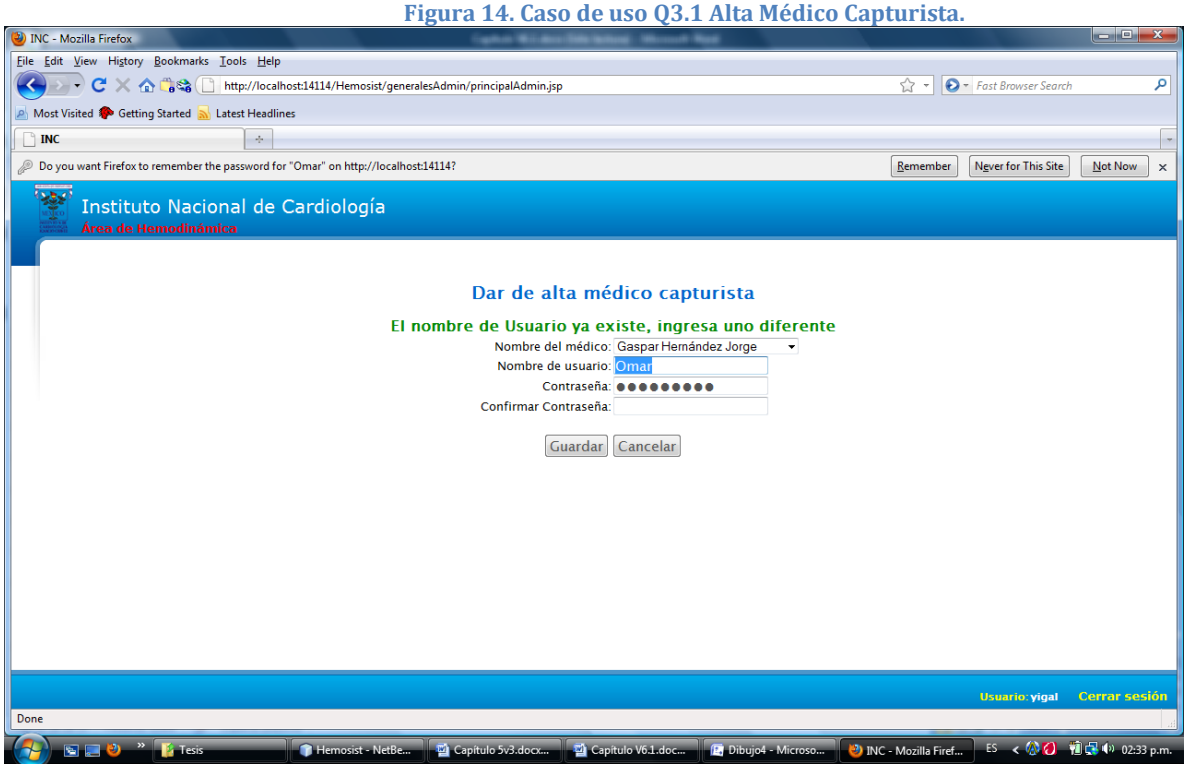

Figura 15. Mensaje al usuario Administrador cuando el usuario ya existe en el caso de uso Q3.1 Alta Médico Capturista.

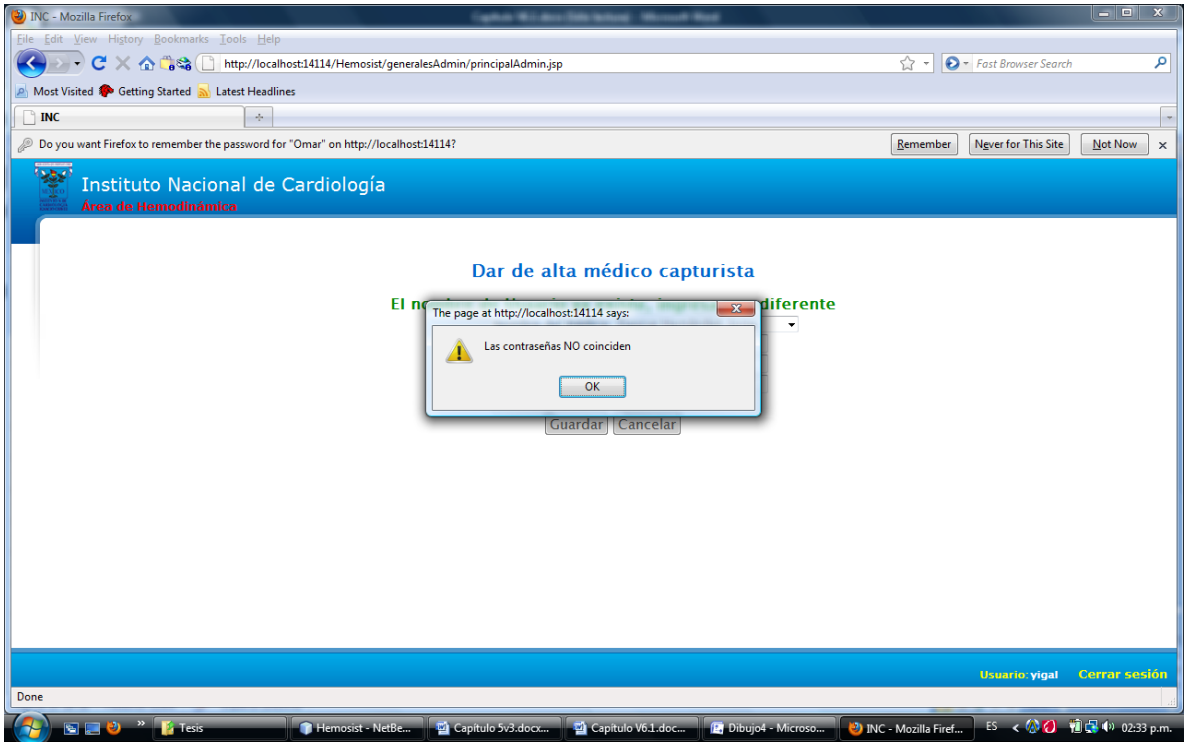

Figura 16. Mensaje al usuario Administrador cuando las contraseñas no coinciden en el caso de uso Q3.1 Alta Médico Capturista.

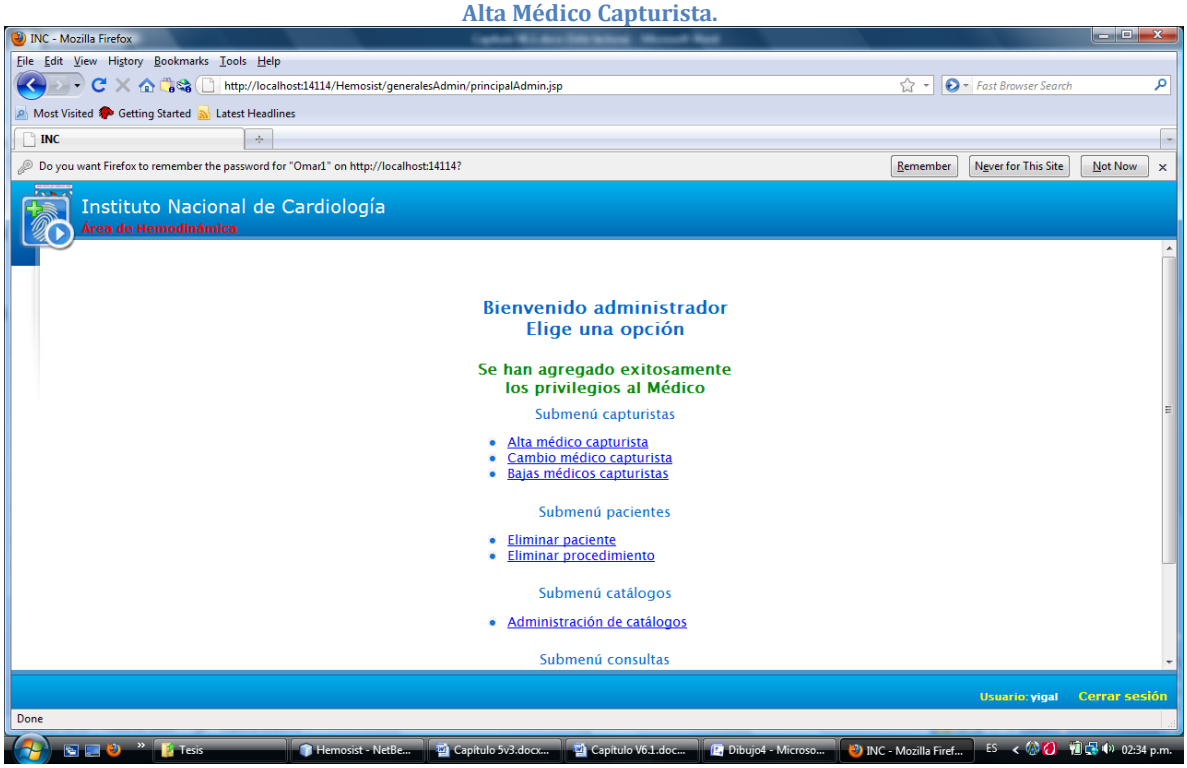

Figura 17. Cuando se han agregado privilegios al médico como usuario Médico Capturista al oprimir el botón Guardar, el sistema manda al caso de uso Q1.1.

La parte del autómata que verifica el caso de uso Q3.2 se muestra en la Figura 18.

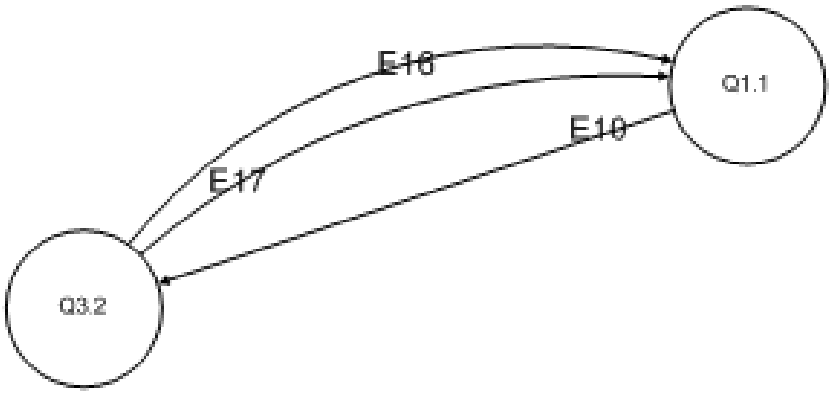

Figura 18. Parte del autómata que prueba el caso de uso Q3.2 Cambio Médico Capturista.

Llegamos al estado Q3.2 al introducir la variable E10 (Cambio médico Capturista). La parte de la tabla, correspondiente al autómata de la Figura 18 se muestra a continuación:

Tabla 11. Tabla de transición de estados para el sistema Hemosist, Cambio Médico Capturista.

Estado final / Estado inicial	Q1.1	Q3.2
Q1.1	-	E10
Q3.2	E16,E17	-

Las variables E16 y E17 (Guardar y Cancelar respectivamente) hacen una transición al estado Q1.1, en caso de que los datos introducidos sean válidos, y si no se muestra el mensaje de error en el mismo estado figuras 15 y 16.

Figura 19. Caso de uso Q3.2 Cambio Médico Capturista.

Figura 20. Cuando se cambia la contraseña al Médico Capturista al oprimir el botón Guardar, el sistema manda al caso de uso Q1.1.

La parte del autómata que verifica el caso de uso Q3.3 se muestra en la Figura 21.

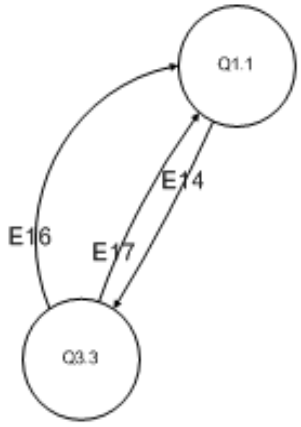

Figura 21. Parte del autómata que prueba el caso de uso Q3.3 Baja Médico Capturista.

Llegamos al estado Q3.3 al introducir la variable E14 (Bajas médico Capturista). La parte de la tabla, correspondiente al autómata de la Figura 21 se muestra a continuación:

Tabla 12. Tabla de transición de estados para el sistema Hemosist, Baja Médico Capturista.

Estado final / Estado inicial	Q1.1	Q3.3
Q1.1	-	E14
Q3.3	E16,E17	-

Las variables E16 y E17 (Guardar y Cancelar respectivamente) hacen una transición al estado Q1.1, en caso de que los datos introducidos fuesen válidos.

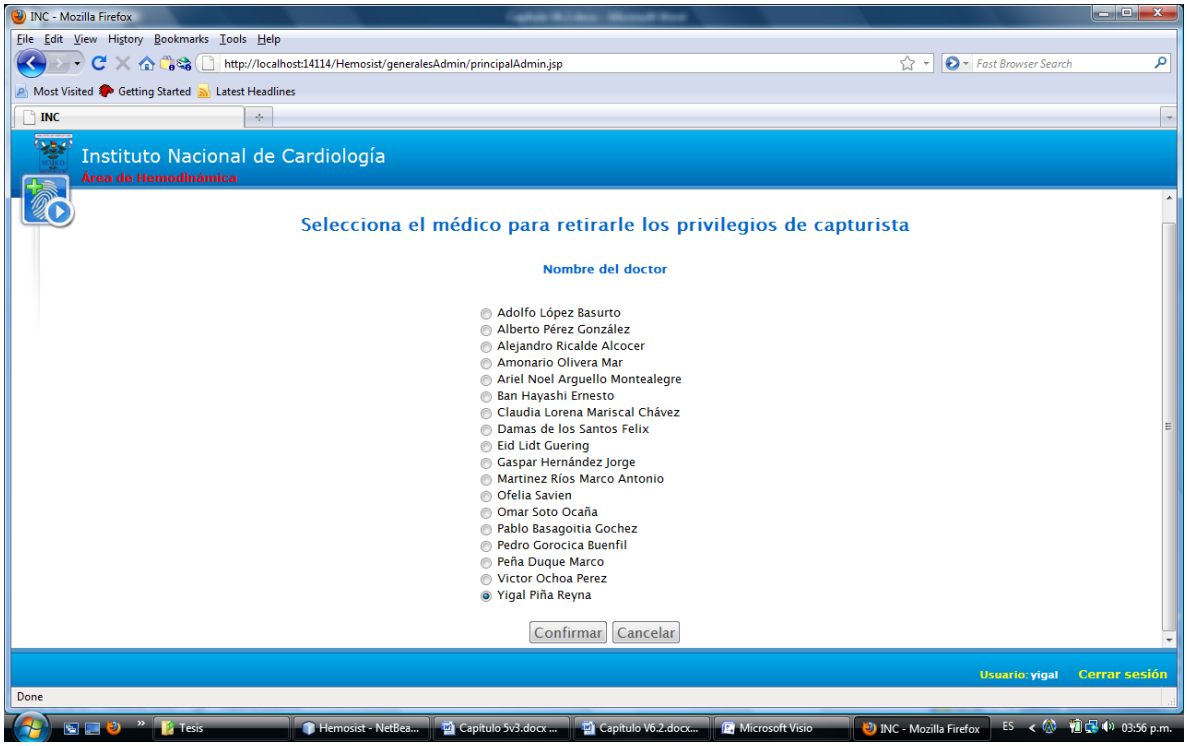

Figura 22. Pantalla del caso de uso Q3.3 Baja Médico Capturista.

Figura 23. Cuando se confirma el dar de baja los privilegios del médico como usuario Médico Capturista al oprimir el botón Confirmar, el sistema manda al caso de uso Q1.1.

- Eliminar paciente: De la base de datos se eliminará a algún paciente, que los usuarios Médicos Capturistas hayan dado de alta previamente. Una vez que se regrese al estado Q1.1 o Q1 habiendo realizado las operaciones designadas se tiene que a partir de dichos estados se llega al estado de aceptación de acuerdo a los primeros casos de prueba (Autenticarse y Salir). La parte del autómata que verifica el caso de uso Q5 se muestra en la Figura 24.

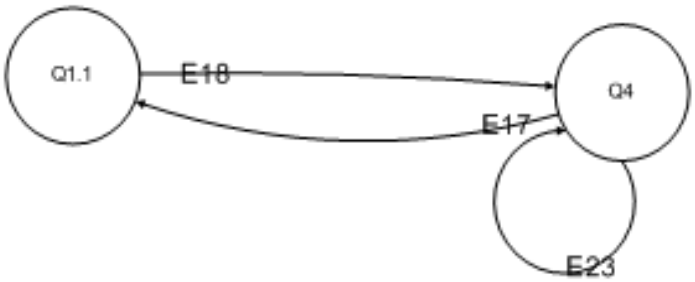

Figura 24. Parte del autómata que prueba el caso de uso Q5 (Eliminar paciente).

Llegamos al estado Q4 a partir del estado Q1.1 al introducir la variable E18 (Eliminar paciente). La parte de la tabla, correspondiente al autómata de la Figura 24 se muestra a continuación:

Tabla 13. Tabla de transición de estados para el sistema Hemosist, Eliminar Paciente.

Estado final / Estado inicial	Q1.1	Q4
Q1.1	-	E18
Q4	E17	E23

La variables E17 (Cancelar) hace una transición al estado Q1.1 y E23 (Confirmar eliminar paciente) hace una transición al estado mismo estado Q4, en caso de haber eliminado al paciente de forma correcta.

Figura 25. Pantalla del caso de uso Q5 Eliminar paciente.

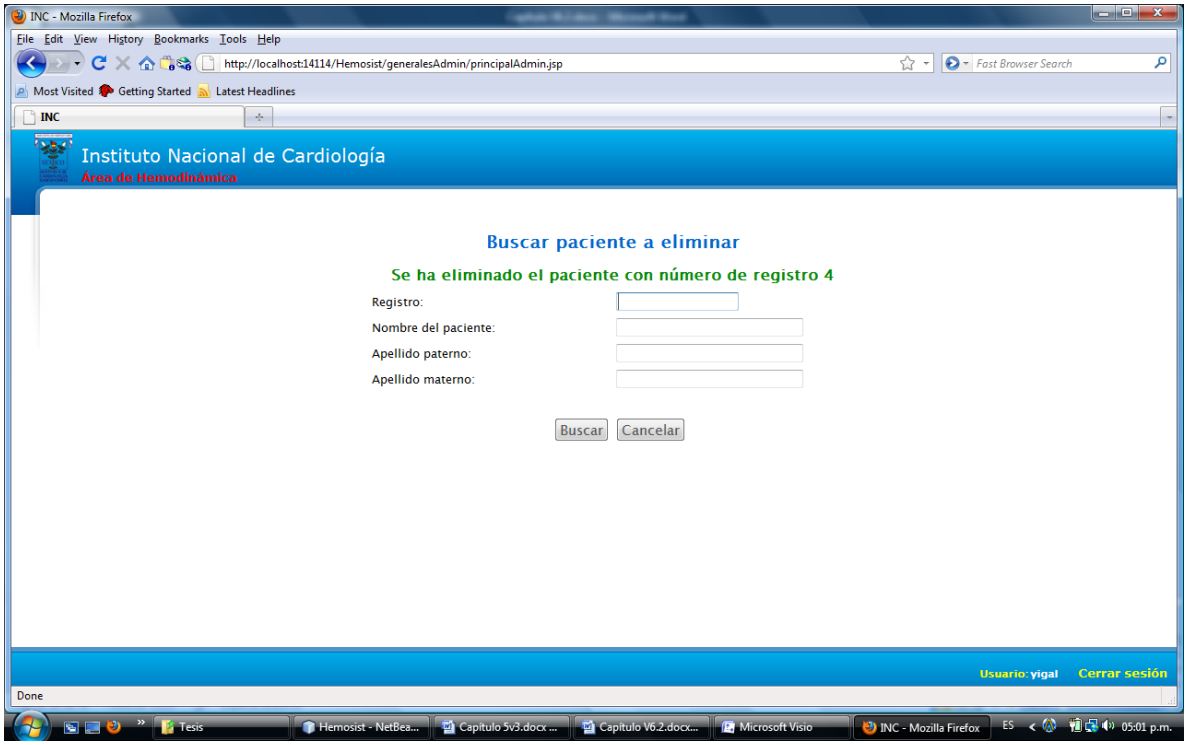

Figura 26. Cuando se elimino un paciente, el sistema se queda en el mismo caso de uso Q5. Cambia de estado hasta introducir la variable E17 (Cancelar).

2. Comparar los resultados de las pruebas con los resultados esperados e investigar los resultados de las pruebas que no coinciden con los esperados. En el sistema Hemosist durante las pruebas que se realizaron sólo el caso de uso Q3.3 (Baja Médico Capturista) no realizaba las operaciones para las que fue planeada.
3. Informar de los defectos a los desarrolladores responsables de los componentes que se cree contiene los fallos para posteriormente puedan ser corregidos. Después de detectar el error encontrado, se procedió a corregir el defecto, se modifico la clase bajaMedicoUsuario.jsp.
4. Informar de los defectos a la persona responsable de las pruebas, quien usará los defectos para evaluar los resultados de realizar dichas pruebas. Se tienen los elementos necesarios para evaluar el resultado de las pruebas en el sistema Hemosist.

7. Evaluar las pruebas.

El análisis de los resultados de las pruebas puede beneficiar a un proyecto de desarrollo como Hemosist de dos maneras diferentes. Como se mencionó en el capítulo uno, la primera es realizar un seguimiento individual de cada uno de los defectos encontrados para que posteriormente sean corregidos. Gracias a lo anterior en el sistema Hemosist se pudieron corregir algunos errores.

Con ese seguimiento se proporciona a la persona responsable del desarrollo del sistema la información necesaria para decir el número de defectos descubiertos, el número de defectos corregidos y el número de defectos a la espera de ser corregidos. En el caso del sistema Hemosist se encontró un defecto al momento de realizar las pruebas en el caso de uso Baja Médico Capturista, ya que la clase bajaMedicoCapturista.jsp tenía un defecto que no lo dejaba cambiar de estado. Con los informes de seguimiento se toman las decisiones sobre el esfuerzo y la conveniencia de corregir todos los defectos conocidos antes de que el desarrollo se declare terminado.

Basándose en el análisis de la tendencia de los defectos Se sugieren tomar las siguientes acciones:

- Las tendencias de defectos encontrados en las pruebas de integración sugieren el continuar con las pruebas de sistema y las pruebas de aceptación.
- Se repararon y corrigieron las partes del sistema que contenían defectos.

Conclusiones

Con base en el desarrollo del presente trabajo de tesis se llegó a las siguientes conclusiones:

Se plantearon las bases teóricas utilizadas para el estudio de las pruebas de software, las cuales abarcan desde el panorama general de ellas, pasando por la clasificación de la granularidad de las mismas, y las actividades que se realizan en el proceso de pruebas como son la planificación, el diseño, la implementación, la realización y el análisis de las pruebas. Se sabe, con base en lo estudiado en el capítulo uno, que las pruebas de software son un elemento muy importante para garantizar la calidad de los sistemas de software. Así se tiene que al efectuar las pruebas al producto final, el sistema de software, debe cumplir con los requerimientos técnicos y empresariales que motivaron su elaboración y que además debe funcionar como se espera que trabaje. Por otra parte se estudiaron los cuatro niveles de pruebas: Pruebas de unidad (que verifican el funcionamiento del software de forma aislada en piezas que son comprobables por separado), pruebas de integración (cuyo propósito es probar que las diferentes unidades trabajen juntas de manera apropiada), pruebas de sistema (que se ocupan del comportamiento del sistema en forma completa) y pruebas de aceptación (que verifican el comportamiento del sistema contra los requerimientos del cliente).

Se realizó un estudio referente al Proceso Unificado, se revisaron lo que son las pruebas de integración en dicho Proceso Unificado y el flujo de trabajo de la prueba para verificar el resultado de la implementación al probar cada construcción.

Se presentaron lo que son las bases teóricas de los conceptos generales de las máquinas de estado, autómatas finitos, representación de autómatas finitos como tabla de transición. Lo anterior para tener una abstracción de lo que se realizó al momento de elaborar la guía que se propuso en el presente trabajo de tesis. Se definieron los elementos básicos para realizar las pruebas de integración con máquinas de estado (estado, transición, evento y salida), el funcionamiento básico para una máquina de estados para sistemas de software y algunas propiedades de los autómatas de estados finitos en las pruebas.

Se identificó que las aplicaciones web, sistemas interactivos en donde el usuario final tiene muchas opciones en el menú, se pueden modelar como máquinas de estados.

Se mezclaron los conceptos de pruebas de software utilizando máquinas de estado y el Proceso Unificado para definir la guía para realizar las pruebas de integración utilizando máquinas de estado siguiendo el Proceso Unificado, complementando el modelo de pruebas propuesto en el capítulo 4 del presente trabajo de tesis, con siete puntos a seguir que son: 1)Definir el objetivo de la prueba, 2) Entender el sistema a probar, 3)Identificar

los estados, 4) Mapear las entradas como eventos que causan las transiciones de estados, 5) Para cada estado de la máquina, definir el conjunto de pruebas y resultados esperados, 6)Aplicar la prueba y 7) Evaluar la prueba.

Se verificó que la guía propuesta fuese de utilidad, por lo cual se realizó la aplicación de dicha guía a un sistema de software que representa un caso real (Hemosist), sistema de software realizado por los alumnos del Posgrado en Ciencia e Ingeniería de la Computación para el departamento de hemodinámica del Instituto Nacional de Cardiología "Ignacio Chávez". Con la generación de un sistema al que se le realizó las pruebas de integración de acuerdo a la guía propuesta.

Gracias al análisis del Proceso Unificado que se efectuó se sabe que durante la fase de inicio se puede hacer parte de la planificación inicial de las pruebas cuando se define el ámbito del sistema. Sin embargo, al observar cómo se desarrollaron las pruebas en el sistema Hemosist, las pruebas se llevan a cabo sobre todo cuando una construcción es sometida a pruebas de integración. Esto quiere decir que la realización de pruebas se centra en las fases de elaboración, cuando se prueba la línea base ejecutable de la arquitectura, y de construcción cuando el grueso del sistema está implementado. Durante la fase de transición el centro se desplaza hacia la corrección de defectos durante los primeros usos y posteriormente a las pruebas de regresión.

Al mezclarse, en el presente trabajo, los conceptos de pruebas de software utilizando máquinas de estado y el Proceso Unificado para definir la guía para realizar las pruebas de integración se tiene que la guía propuesta apoya a la elaboración del sistema de software en la parte de las pruebas de la siguiente forma:

1) Definir el objetivo de la prueba, al definirse el alcance y planear el utilizar los recursos asignados a la elaboración de las pruebas se maximiza el retorno de inversión probando la parte del software que consideremos importante y se aprovechan los recursos asignados. Con base en la plantilla propuesta en la página 37 se planifican los esfuerzos de las pruebas, anotando los requerimientos y los riesgos, así como las limitaciones que se tienen al planificar las pruebas; los responsables, los datos de prueba y el ambiente en el que se realizarán las mismas. Esto ayuda a comprender hasta dónde se quiere llegar en cuanto a las pruebas se refiere y que recursos tenemos para llegar a esa meta.

2) Entender el sistema a probar, el tener una representación de las funcionalidades del sistema es un requerimiento importante para poder construir cualquier modelo de prueba es por esto que la plantilla de la página 38 ayuda a determinar cuáles son los casos de uso a probar, lo documentación que se tiene a la mano, los usuarios del sistema, la documentación aplicable a cada entrada y las condiciones bajo las que se produce las respuestas a las entradas. En general este punto es importante ya que para poder modelar

pruebas se necesita entender que es lo que se quiere probar; por lo que el punto 2 de la guía proporciona una herramienta para entender el sistema a probar de cualquier proyecto.

3) Identificar los estados, el primer paso para definir como tal las pruebas en la guía es tomar en cuenta los casos de uso y realizar un mapeo para tener los estados del sistema; en esta parte se tienen los elementos que se quieren probar. Es importante considerar los estados del sistema así como sus entradas, en el sistema Hemosist esta parte representa el poder mapear cada uno de los casos de uso identificados con los estados que modelan el sistema.

4) Mapear las entradas como eventos que causan las transiciones de estados, es importante el entender que no todas las entradas de un sistema causan una transición a otro estado, otro caso de uso, por lo que este punto en la guía es importante en la parte de interactuar un caso de uso con otro dentro del sistema; en el sistema Hemosist, y en cualquier otro sistema de software de tipo interactivo, los casos de uso podían funcionar muy bien de forma individual, pero al momento de interactuar con otro caso de uso en ocasiones se encontraron problemas que no se habían tomado en cuenta al momento de trabajar un caso de uso en forma individual.

5) Para cada estado de la máquina, definir el conjunto de pruebas y resultados esperados, la tabla propuesta en la página 41 que incluye todas las entradas que causan alguna transición; con ella se debe presentar el estado inicial del sistema, pantalla de bienvenida en el caso particular del sistema Hemosist, el evento que se va a probar, el caso de uso que se está probando y el estado que al final de realizar determinadas operaciones debe llegar el sistema.

6) Aplicar la prueba, el realizar las pruebas de forma informal no sería de utilidad al momento de evaluar si la inversión que se realizó para llevarla a cabo dio frutos o si se siguió una metodología adecuada para llevar a cabo dichas pruebas; por lo que el tener una forma sistematizada y ordenada de realizar las pruebas de integración le puede decir a la persona encargada de realizar las pruebas si su trabajo está enfocado no sólo a detectar y corregir defectos sino a también a evaluar si se tiene un producto con la calidad necesaria para ser liberado teniendo en cuenta que se están realizando las pruebas de una forma adecuada.

7) Evaluar la prueba, el evaluar las pruebas como se mencionó es el de valorar si satisface el objetivo de la prueba para determinar si el software cumple con la calidad suficiente para posteriores pruebas. Basándose en la evaluación de la tendencia de los defectos la persona responsable de las pruebas puede sugerir otras acciones como realizar pruebas adicionales, relajar el criterio para las pruebas o aislar las partes del sistema que tengan una calidad aceptable para ser entregadas como parte del producto final.

Por otra parte en el presente trabajo se verificó que la guía propuesta fuese de utilidad, por lo cual se realizó la aplicación de dicha guía a un sistema de software que representa un caso real (Hemosist), por lo que dicha prueba se evaluó con el sistema de software realizado por los alumnos del Posgrado en Ciencia e Ingeniería de la Computación para el departamento de hemodinámica del Instituto Nacional de Cardiología "Ignacio Chávez". Con la generación de un sistema al que se le realizó las pruebas de integración de acuerdo a la guía propuesta. En el caso del sistema Hemosist se encontró un defecto al momento de realizar las pruebas en el caso de uso Baja Médico Capturista, ya que la clase bajaMedicoCapturista.jsp tenía un defecto que no lo dejaba cambiar de estado. Con los informes de seguimiento se toman las decisiones sobre el esfuerzo y la conveniencia de corregir todos los defectos conocidos antes de que el desarrollo se declare terminado.

Así tenemos que la guía que se definió para realizar las pruebas de integración en sistemas de software de tipo interactivo utilizando máquinas de estado sirve para realizar las pruebas de una forma sistematizada en la que se toman en cuenta el objetivo de la prueba, que se entienda a dónde que es lo que debe realizar el producto final, los casos de uso que componen el sistema así como sus entradas y la interacción de éstas con los distintos casos de uso. Todo esto con lo que propone el Proceso Unificado, Planear las pruebas, Diseñar e implementar dichas pruebas y realizar las pruebas y finalmente el analizar los resultados.

Basándose en el análisis de la tendencia de los defectos se sugieren tomar las siguientes acciones:

- Las tendencias de defectos encontrados en las pruebas de integración sugieren el continuar con las pruebas de sistema y las pruebas de aceptación.
- Se repararon y corrigieron las partes del sistema que contenían defectos.

Contribuciones

Por todo lo anterior se puede decir que la guía propuesta puede ser utilizada en sistemas de software de tipo interactivos para detectar y corregir errores de tipo interactivos en las pruebas de integración para dichos sistemas.

Con base en la guía propuesta se puede desarrollar una suite de pruebas utilizando máquinas de estado desde un modelo de análisis o modelo de diseño en forma efectiva y temprana para verificar el comportamiento de un subsistema. Con la guía propuesta de máquina de estado se pueden realizar las pruebas de integración sobre muchos alcances del sistema, ya que soporta explícitamente el desarrollo de casos de prueba basados en hilos.

Utilizando la guía propuesta de pruebas con máquinas de estado se tiene una visión general relacionada con el comportamiento completo del sistema que se quiere poner a prueba. Con una terminología exacta y el uso adecuado de los métodos formales, se puede

escalar el proceso de prueba, lo que justificará los gastos que pueda generar y ajustarse con el presupuesto que se tiene destinado para realizar las pruebas.

Una contribución es que se presentó un trabajo previo en el Coloquio Nacional de Investigación en Ingeniería de Software y Vinculación Academia-Industria (CoNIIS, 2010) realizado del 29 de septiembre al 1ro de octubre del 2010 con buenas críticas.

Trabajo futuro

Como trabajo futuro se propone el automatizar las pruebas a la hora de implementarlas, en los casos que sean posibles, ya que en ocasiones se procesan grandes cantidades de datos de entrada para ser probados y producen grandes cantidades de datos de salida como resultado de las pruebas y por tanto será de gran utilidad poder visualizar los datos en forma clara para que puedan analizarse correctamente y los resultados de las pruebas puedan ser interpretados de manera correcta con grandes cantidades de datos.

Bibliografía

A.S.Krishnakumar, K.-T. C. (1993). Automatic Functional Test Generation Using The Extended Finite State Machine Model. *Annual ACM IEEE Desig Automation Conference* , 86-91.

Belli, F. (2001). Finite State Testing and Analysis of Graphical User Interfaces. *IEEE Comp. Press* , 3e4-43.

Binder, R. (2005). *Testing object-oriented systems: models, patterns, and tools.* EEUU: Adisson-Wesley.

Brookshear, J. G. (1989). *Teoría de la computación, Lenguajes formales, autómatas y complejidad.* México: Addison-Wesley Iberoamericana.

Cem Kaner, J. B. (2002). *Lessons Learned in Sofware Testing.* Wiley Computer Publish.

El-Far, I. K. (2001). Model-based Software Testing. En J. Marciniak, *Encyclopedia on Software Engineering.* United States: Wiley.

es.wikipedia.org. (7 de 8 de 2010). Obtenido de es.wikipedia.org:

Fujiwara, S. G. B. (s.f.). Test selection based on finite state models.

http://es.wikipedia.org/wiki/Aut%C3%B3matas_finitos

Hopcroft, J. E., & Ullman, J. D. (1993). *Introducción a la teoría de autómatas, lenguajes y computación.* México: Compañía editorial Continental.

Hyoung Seok Hong, Y. R. (s.f.). Testing of Object-Oriented Programs Based on Finite State Machines.

IEEE. (1998). Software Engineering Technical Committee. *IEEE Standard for Software Test Documentation* .

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process.* Addison-Wesley Professional.

Jacobson, I. G. B. (2000). *El Proceso Unificado de desarrollo de software.* Madrid: Pearson Educación.

Larman, C. (2003). *Agile and Iterative Development: A Manager's Guide.* Addison Wesley.

Lee, D. (s.f.). Principles and methods of testing finite state machines.

Neil Walkinshaw, K. B. (2007). Automated discovery of state transitions and their functions in source code. *Software Testing, verification and reliability*.

Nogueira de Lucena, F., & Liesenberg, H. K. (1996). Introducción a los Statecharts. *Soluciones avanzadas* , 46-54.

Rumbaugh, J. M. B. (1995). *Modelado y Diseño Orientado a Objetos, Metodología OMT. Primera edición en español.* España: Prentice Hal.