*Universidad de las Ciencias Informáticas*
*Facultad 1*


# *Título: Procedimiento de pruebas para aplicaciones en tarjetas inteligentes*

***Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas***

***Autor:*** José Andrés Mariño Arango

***Tutor:*** *Ing. Ander Sánchez Jardines*

*La Habana, Cuba*
*"Año 54 de la Revolución"*

**“Dentro de la Revolución, todo; contra la Revolución, nada”**

# DECLARACIÓN DE AUTORÍA

**MINISTERIO DE EDUCACIÓN SUPERIOR**

**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

Declaro ser el único autor del presente trabajo de diploma y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo. Autorizo a dicho centro para que haga el uso que estime pertinente con este trabajo de diploma.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2012.

_____

Autor: José Andrés Mariño Arango

_____

Tutor: Ing. Ander Sánchez Jardines

# DEDICATORIA

## Dedicatoria

# AGRADECIMIENTOS


## Agradecimientos

# RESUMEN

El factor fundamental para el éxito en la producción de *software* es la calidad y para ello es necesario que el cliente se encuentre satisfecho con el producto desarrollado. La realización de las pruebas de *software* constituye un factor esencial para garantizar la calidad del producto, implicando que se debe cumplir con todas las exigencias y perspectivas del cliente. La siguiente investigación se centra en crear un procedimiento que permita realizar pruebas con herramientas automatizadas a las aplicaciones que se desarrollan en el Departamento de Tarjetas Inteligentes en el Centro de Identificación y Seguridad Digital. Como resultado, al automatizar dichas pruebas el proceso será más rápido y flexible debido a que actualmente se realizan las pruebas manualmente. En el procedimiento se utilizan las buenas prácticas de la metodología XP y se realizan las pruebas automatizadas con las herramientas *NetBeans* 7.1 y *JTharness*-4.4. Estas fueron seleccionadas después de un estudio abarcador de numerosas herramientas. Con el procedimiento presentado se gana en eficiencia y organización.

**Palabras clave:** calidad, cliente, herramientas, pruebas, *software*, tarjetas inteligentes.

# ÍNDICE

## 1 Índice de contenido

# ÍNDICE

# ÍNDICE

# ÍNDICE

# ÍNDICE

## Índice de Figuras

# ÍNDICE

## Índice de Tablas

# INTRODUCCIÓN

El vertiginoso desarrollo tecnológico en la empresa del *software* mundialmente ha marcado un gran hito en la sociedad, pasando a lo que muchos denominan como era electrónica. Esto ocurre debido al incremento de mejoras en los diferentes servicios, desde aplicaciones informáticas utilizadas por instituciones y empresas privadas hasta la creación de artefactos para el uso doméstico y/o personal. Dicha mejora en los servicios hace más cómodo y efectivo el trabajo para alcanzar numerosos beneficios al hombre. Por la competencia intrínseca que lleva esta revolución informática, teniendo en cuenta los fallos de *software* que en ocasiones afecta a las empresas de este tipo, de manera parcial o crítica, reflejados en retrasos respecto a la fecha de entrega del producto, pérdidas de tiempo o económicas, problemas ambientales o sociales, etc, las mismas no dirigen sus estrategias de producción solamente a desarrollar *software*, sino que incluyen como parte de su política productiva, procesos para la gestión de la calidad de sus productos, con el objetivo de alcanzar la satisfacción del cliente, al ser este el eslabón principal en la cadena de producción, distribución y consumo de las actuales actividades económicas que vaya a realizar cualquier industria de *software*. En la mejora continua de los procesos de desarrollo de *software* se utilizan diferentes metodologías y estándares de certificación de los procesos de desarrollo de *software*. Sin importar cual metodología sea utilizada por un equipo de desarrollo, las pruebas realizadas sobre el producto, reflejan si los objetivos planteados por el cliente son satisfechos por el producto desarrollado.

Entre las tecnologías que han provocado un gran impacto en la sociedad se encuentran las denominadas tarjetas inteligentes, sobre las cuales se basan las soluciones de diferentes tipos de aplicaciones. Cuba, aunque se encuentra fuertemente bloqueada por el imperialismo y sus recursos son limitados, a la hora de la adquisición de *software* y *hardware* hace todo lo posible para estar a la altura de los países desarrollados llevando a cabo la informatización de todos los sectores del país. En el 2002 se crea en Cuba la Universidad de las Ciencias Informáticas (UCI), con la cual se logró aumentar el desarrollo de la industria del *software*, vinculándola a las necesidades del desarrollo del país y aumentando la presencia de sus productos en su perfil exportador. En esta existen varios centros de producción de *software* entre los cuales se encuentra el Centro de

# INTRODUCCIÓN

Identificación y Seguridad Digital (CISED). En el CISED se encuentra el Departamento de Tarjetas Inteligentes donde se desarrollan productos y servicios utilizando, como su nombre lo indica, tarjetas inteligentes y se brinda asesoría técnica a proyectos relacionados con estas tecnologías.

**Situación problemática:** Las aplicaciones que se desarrollan en dicho Centro para tarjetas inteligentes, tienen características particulares que las hacen diferentes a la mayoría de las soluciones que se desarrollan en la UCI. Las aplicaciones en tarjetas inteligentes necesitan un lector y una tarjeta que se comunican mediante protocolos de comunicación. Actualmente el Departamento de Tarjetas Inteligentes no cuenta con un procedimiento de pruebas para garantizar la calidad de sus aplicaciones. Por lo que se realizan pruebas manuales y sin una adecuada organización. Para mejorar la calidad de las aplicaciones del Departamento, se hace necesario desarrollar un procedimiento de pruebas con herramientas para automatizar dichas pruebas. Con este procedimiento se lograría un control exhaustivo de las fallas que se detectan en dichas aplicaciones durante las etapas de las pruebas. Conociendo las fallas presentadas se pueden tomar acciones correctivas; por lo que se definió el siguiente **problema científico de la investigación:** ¿Cómo mejorar la calidad durante el desarrollo de las aplicaciones en tarjetas inteligentes en el Departamento de Tarjetas Inteligentes perteneciente al Centro de Identificación y Seguridad Digital? Se define como **objeto de estudio:** Procedimientos de pruebas de software y el **campo de acción** se enmarca en: Procedimientos de pruebas de software para aplicaciones en tarjetas inteligentes.

**Objetivo general:** Definir un procedimiento de pruebas para el desarrollo de aplicaciones en tarjetas inteligentes en el Departamento de Tarjetas Inteligentes del Centro de Identificación y Seguridad Digital, con vista a lograr un mayor control de las deficiencias que se presenten.

Los **objetivos específicos** que se derivan del objetivo general son:

- ✓ Elaborar el marco teórico de la investigación.
- ✓ Diseñar el procedimiento de pruebas a aplicar a las aplicaciones en tarjetas inteligentes.
- ✓ Validar de forma teórica y práctica el procedimiento de pruebas propuesto.

# INTRODUCCIÓN

Para dar solución al problema existente se tiene la siguiente **idea a defender:** Si se propone un procedimiento de pruebas para aplicaciones en tarjetas inteligentes entonces se puede lograr un mayor control de las deficiencias y se reducirá el tiempo invertido en detectar fallas internas y externas.

Con los objetivos trazados se plantea el cumplimiento de las siguientes **tareas de la investigación:**

- ✓ Análisis de las características lógicas y físicas de las tarjetas inteligentes.
- ✓ Caracterización de las aplicaciones que se desarrollan para tarjetas inteligentes.
- ✓ Caracterización de los tipos de pruebas de software existentes.
- ✓ Definición de las actividades, roles involucrados, artefactos y herramientas a utilizar en el procedimiento de pruebas para aplicaciones en tarjetas inteligentes.
- ✓ Aplicación del procedimiento de pruebas propuesto en el desarrollo de una aplicación desarrollada para tarjetas inteligentes.
- ✓ Validación del procedimiento de pruebas propuesto utilizando el método experto Delphi.

**Estrategia de la investigación:**

En el desarrollo del trabajo se pondrán en práctica varios métodos que facilitarán las tareas de la investigación.

Los **métodos científicos** a utilizar son:

**Métodos teóricos:**

- ✓ **Método Analítico - Sintético:** Durante todo el proceso investigativo se realizará un estudio a profundidad de toda la bibliografía relacionada con el tema y a partir del análisis realizado se seleccionará una síntesis de lo estudiado.
- ✓ **Método Histórico - Lógico:** Para la realización de la investigación se hará necesario estudiar la evolución del problema y la existencia de procedimientos de pruebas similares al que se pretende elaborar.

- ✓ **Modelación:** Este método permitirá la creación de modelos (propuestas, alternativas, estrategias, etc.) con vistas a investigar la realidad.

**Métodos empíricos:**

- ✓ **Observación:** Consiste en la percepción planificada dirigida a un fin y relativamente prolongada de un hecho o fenómeno. Este método permitirá percibir directamente cómo se hacen las pruebas de calidad en la práctica para así poder llevar esos conocimientos al tema propuesto.
- ✓ **Encuesta:** Durante la realización de la investigación será necesario realizar encuestas para la validación de la solución propuesta. A través de este método se conocerá el criterio de los expertos en la clasificación de la solución en desarrollo que se podrá llevar a la práctica en caso de ser aceptada.
- ✓ **Entrevista:** Se realizarán a especialistas en calidad de *software* para conocer cómo se debe llevar a cabo el procedimiento y a especialistas en desarrollo de *software* de Tarjetas Inteligentes ya que ellos están en contacto directo con el *software,* constituyendo la entrevista un medio para el conocimiento cualitativo de los fenómenos.

**Actualidad y necesidad del trabajo:**

El Centro de Identificación y Seguridad Digital, dentro de su estrategia y plan futurista, pretende seguir desarrollando *software* para tarjetas inteligentes por tanto el trabajo será un modelo útil, organizado y patentado por el cual guiarse en la realización de las pruebas a dichos *software*.

**Resultados Esperados:**

- ✓ Definición de un procedimiento de pruebas adaptado a las características de las aplicaciones que se desarrollan para aplicaciones inteligentes, que facilite el control de la calidad de las mismas por el equipo de desarrollo.
- ✓ Evaluación del procedimiento propuesto a través del método Delphi y su aplicación en una de las aplicaciones desarrolladas por el Departamento de Tarjetas Inteligentes del Centro de Identificación y Seguridad Digital.

# INTRODUCCIÓN

**Estructuración del contenido y una breve explicación de sus partes**

**Capítulo1: Fundamentación teórica**

En este capítulo se proporciona una panorámica general acerca de las características del *software* de tarjetas inteligentes; de manera que se revisará su estructura y funcionamiento, además se incluirá una descripción de los diferentes tipos de pruebas aplicadas a este *software*.

**Capítulo 2: Procedimiento propuesto**

En este capítulo se proponen un conjunto de pruebas a aplicar en el desarrollo del *software* y las herramientas a utilizar en el procedimiento, exponiéndose ejemplos, información y datos.

**Capítulo 3: Validación del procedimiento de pruebas**

En este capítulo se valida la propuesta del procedimiento de pruebas mediante la utilización del método experto Delphi y se aplica en un *applet* del Departamento de Tarjetas Inteligentes.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## 1.1 Introducción

En este capítulo se hará un estudio de las características físicas y lógicas de las tarjetas inteligentes y de los diferentes tipos de *software* existentes en el mundo de las tarjetas inteligentes contando con numerosos ejemplos de aplicaciones en tarjetas inteligentes. También contará este capítulo con una caracterización de las metodologías existentes y de los diferentes enfoques de diseños de pruebas, presentando además ejemplos de procedimientos, modelos y tipos de pruebas, realizadas a distintas aplicaciones informáticas nacionales e internacionales.

## 1.2 Tarjetas Inteligentes

### 1.2.1 Definición de Tarjeta Inteligente

El término “Tarjetas Inteligentes” ha sido utilizado de manera no muy precisa en algunas fuentes de información al referirlas como aquellas tarjetas que tienen la capacidad de relacionar información con alguna aplicación en particular, como por ejemplo las tarjetas de barra magnética, las tarjetas de barra óptica, las tarjetas con chip de memoria y las tarjetas con microprocesador. Sin embargo es más preciso utilizar este término para aquellas que tienen chip con memoria y para las que tienen chip con microprocesador.

Entre algunas definiciones de las tarjetas inteligentes se encuentran las siguientes:

“Una tarjeta inteligente es un tipo de tarjeta de plástico con una computadora de circuito integrado (chip) empotrado que almacena y realiza transacciones de datos entre usuarios”

“Sistema portador de información electrónico que usa tarjetas de plástico del tamaño de una tarjeta de crédito con un circuito integrado incrustado que guarda información de los procesos”

“Una tarjeta inteligente es una computadora portátil, resistente a daños, con un almacén de datos programable. Es de forma y tamaño exacto al de una tarjeta de crédito, puede

almacenar 32KB o más información sensible y también puede realizar procesamiento moderado de datos”

Son muchas las definiciones que se pudieran encontrar para las tarjetas inteligentes. Sin embargo, la mayoría de ellas se centran en una idea principal: la de contener incrustado un chip de circuito integrado que le permite almacenar información de manera segura e incluso procesar esta información. Tomando como base a las anteriores, se propone la siguiente definición:

“Una tarjeta inteligente es un dispositivo que tiene las características físicas similares a las de una tarjeta de crédito convencional y que además tiene un circuito integrado empotrado, con memoria y capacidades de procesamiento de información, que le permite ejecutar aplicaciones para almacenamiento y transferencia de información en forma segura, confiable y eficiente” (2).

### 1.2.2 Antecedentes

Las primeras tarjetas surgieron en el año 1950, en aquel entonces eran de material plástico. En 1968 los alemanes Jürgen Dethloff y Helmut Gröttrup desarrollaron un sistema de identificación, que básicamente es una tarjeta con un circuito integrado.
Dos años más tarde en 1970 el japonés Arimura Ichiro realizó algunos aportes importantes al desarrollo de la tarjeta con circuito integrado en la integración lógica, aritmética y almacenamiento en el chip. Después de varios intentos de patentes en 1974 el inventor francés Roland Moreno patentó la primera tarjeta con circuito integrado, que llamó "Sistema de transferencia de datos" y que explicaba el funcionamiento de una tarjeta que disponía de una memoria para almacenar información, debido a esto consideran a Roland el padre de la tarjeta con Circuito Integrado. En 1987 se expiden los primeros estándares Internacionales para las tarjetas con circuito integrado (ISO 7816 – establece los parámetros de fabricación de las tarjetas inteligentes).
Dentro de sus aplicaciones se encuentra el primer monedero electrónico en 1997 y las primeras “tarjetas Java” en 1998 (2).

### 1.2.3 Tipos de Tarjetas Inteligentes

**De memoria**

**Características**

Portadora de datos, accesibles mediante protocolo síncrono, su transmisión es vulnerable y pueden estar cifradas.

Existen 3 tipos de tarjetas de memoria estos son:

- ✓ Común, solamente almacenan datos y no tienen capacidades de procesamiento.
- ✓ Segmentada / Protegida, incluyen lógica de control para el acceso a la memoria.
- ✓ Con valor almacenado, están diseñadas para el propósito específico de almacenamiento de valores o llaves (2).

Los componentes que contienen son:

- ✓ EEPROM[1]
- ✓ ROM[2]
- ✓ Lógica de seguridad integrada (1)

**Con microprocesador**

**Características**

Portadora de datos y realizadora de tareas entre otras como leer, escribir y cifrar. Presenta interacción entre tarjeta y computadora, ejemplo:

- ✓ Autorización de tarjetas para un sistema determinado.
- ✓ Autenticación de usuarios.
- ✓ Credenciales de tarjetas/computadoras para realizar transacciones.

Los componentes que contienen son:

- ✓ EEPROM
- ✓ ROM
- ✓ RAM[3]
- ✓ CPU[4]
- ✓ Lógica de seguridad integrada

**Tarjetas con chip que sirven para el mecanismo de acceso**

[1] (Electrical Erasable Programmable Read Only Memory): Memoria de almacenamiento.
[2] (Read Only Memory): Memoria persistente en la que se establece el sistema operativo de la tarjeta.
[3] (Random Access Memory): Es la memoria volátil con que trabaja el procesador.
[4] (Central Processing Unit): Es el procesador de la tarjeta.

**Características**

Existen 3 tipos de tarjetas con chip estas son:

- ✓ De contacto, que presenta inserción física, está sujeta a desgaste y rasgadura.
- ✓ Sin contacto, presenta frecuencia de radio lo cual le da rapidez y es resistente al desgaste, la suciedad u otros elementos que puedan entorpecer la tarjeta.
- ✓ *Vault*, para dispositivos magnéticos y de chip.

Las tarjetas de contacto y sin contacto pueden ser de interfaz doble (presenta un solo chip) y combinadas (presentan dos chips).

**Tarjetas según su tamaño:**

- ✓ Tarjeta: tarjeta de banda magnética.
- ✓ Mini tarjeta: billete con cinta magnética.
- ✓ Módulo: tamaño mínimo para albergar chip y sus contactos.

**Tarjetas según su forma:**

- ✓ Tarjeta
- ✓ Testigo USB
- ✓ Anillo
- ✓ Llave (1)

### 1.2.4 Estructura de una Tarjeta Inteligente

Una tarjeta inteligente contiene un microprocesador de 8 Bytes con su CPU, su RAM y su ROM, su forma de almacenamiento puede ser EPROM o EEPROM. El programa ROM consta de un sistema operativo que maneja la asignación de almacenamiento de la memoria, la protección de accesos y maneja las comunicaciones. El sendero interno de comunicación entre los elementos (BUS) es totalmente inaccesible desde afuera del chip de silicona por ello la única manera de comunicar está totalmente bajo el control del sistema operativo y no hay manera de poder introducir comandos falsos o requerimientos inválidos que puedan sorprender las políticas de seguridad (3).

### 1.2.5 Protocolos de comunicación con tarjetas inteligentes

**APDU**

Unidad de datos de protocolo de aplicación (*Application Protocol Data Unit*), es la unidad de comunicación entre un lector y una tarjeta. Su estructura está definida en el estándar ISO 7816, existiendo dos tipos de categorías de APDU, *APDU Command* (Comando APDU) y *APDU Response* (APDU Respuesta) (3).

***GlobalPlatform***

*GlobalPlatform es* una organización independiente enfocada a gestionar una infraestructura estandarizada para el desarrollo y despliegue de tarjetas inteligentes. Proporciona un conjunto de especificaciones universalmente reconocidas e implementadas, junto con configuraciones de mercado, aplicación de esas especificaciones y documentos de apoyo. Cubriendo toda la infraestructura de tarjetas inteligentes (las tarjetas, dispositivos y sistemas) estos documentos técnicos ofrecen una plataforma tecnológica dinámica y completa para el desarrollo de programas de tarjetas inteligentes, para poder establecer una conexión segura con la misma y administrar sus aplicaciones.

Las tarjetas, dispositivos y sistemas *GlobalPlatform*, son interoperables, independientemente de la tecnología del proveedor y la flexibilidad de su infraestructura técnica, garantizan que pueda responder a las necesidades básicas en el instante del despliegue inicial. Ofreciendo a los emisores la seguridad de que la infraestructura que han elegido será capaz de adaptarse y crecer a medida que cambian las condiciones de negocios (3).

**Estándar ISO/IEC- 7816**

Todas las Tarjetas Inteligentes cumplen los estándares de la serie ISO 7816. El objetivo de estos es lograr la interoperabilidad entre distintos fabricantes de tarjetas y lectores de las mismas, en lo que respecta a características físicas, comunicación de datos y seguridad. Estos estándares son basados en los ISO 7810 e ISO 7811, los cuales definen características físicas de tarjetas de identificación. Las características de comunicación de las tarjetas sin contacto son definidas en estándares como el ISO/IEC 14443.

Descripción de algunas partes del estándar ISO 7816:

- ✓ 7816-1: Características físicas.
- ✓ 7816-2: Dimensiones y ubicaciones de los contactos.
- ✓ 7816-3: Señales electrónicas y protocolo de transmisión.

- ✓ 7816-4: Comandos de intercambio inter-industriales.
- ✓ 7816-5: Sistema de numeración y procedimientos de registro.
- ✓ 7816-6: Elementos de datos inter-industriales.
- ✓ 7816-7: Comandos inter-industriales y consultas estructuradas para una tarjeta.
- ✓ 7816-8: Comandos inter-industriales relacionados con seguridad.
- ✓ 7816-9: Comandos adicionales inter-industriales y atributos de seguridad.
- ✓ 7816-10: Señales electrónicas y respuesta para reiniciar una tarjeta inteligente síncrona (3).

**Estándar ISO/IEC 14443**

ISO 14443 es un estándar internacional relacionado con las tarjetas inteligentes sin contacto gestionado conjuntamente por la Organización Internacional de Normalización (ISO) y Comisión Electrotécnica Internacional (IEC).

El estándar ISO 14443 consta de cuatro partes y se describen dos tipos de tarjetas: tipo A y tipo B. Las principales diferencias entre estos tipos son los métodos de modulación, codificación de los planes (parte 2) y el protocolo de inicialización de los procedimientos (parte 3). Las tarjetas de ambos tipos (A y B) utilizan el mismo protocolo de alto nivel (llamado T=CL) que se describe en la parte 4. El protocolo T=CL especifica los bloques de datos y los mecanismos de intercambio.

- ✓ ISO/IEC 14443-1: Características físicas.
- ✓ ISO/IEC 14443-2: Energía de radiofrecuencia y señal de interfaz.
- ✓ ISO/IEC 14443-3: Inicialización y anticolisión.
- ✓ ISO/IEC 14443-4: Protocolo de transmisión (3).

## 1.3 Metodologías de desarrollo de software

Desde comienzos de los años cincuenta con la aparición de los primeros sistemas de información, han surgido diferentes métodos, paradigmas y modelos de procesos con la intención de soportar y ser capaces de manejar los esfuerzos puestos en el desarrollo de

los mismos. Sin embargo, algunos métodos de desarrollo en lugar de lograr flexibilidad y ser más livianos, se tornaron tediosos en cuanto a la documentación y a los procesos que los desarrolladores debían respetar. A éstos modelos, debido a lo indicado anteriormente, se les llamó pesados o métodos tradicionales. Un ejemplo de ellos es el análisis y diseño estructurado.

Con el paso del tiempo, la evolución rápida de la industria y la tecnología indicaron a las empresas que sus sistemas debían poder adaptarse a ellos. Así fue como se dieron cuenta de que, siguiendo bajo las metodologías tradicionales les serían muy complejos y hasta veces imposible cumplir con todos los procedimientos, artefactos y procesos que estas requerían. Como consecuencia de esto e intentando dar respuesta a éste escenario las consultoras comenzaron a desarrollar métodos y buenas prácticas que les permitiesen poder adaptarse a los cambios. Este fue el inicio de las metodologías ágiles, que se componen de un conjunto de técnicas que comparten principios básicos (4).

### 1.3.1 Modelos de procesos de desarrollo de software

Las modelos de procesos que se detallarán en esta sección son: proceso en cascada, desarrollo iterativo y proceso espiral.

#### 1.3.1.1 Proceso cascada

Fue introducido por en 1970 y luego se hizo una extensión de este modelo agregando pasos adicionales. Una de las contribuciones de este modelo fue la creación de la cultura del pensar antes de codificar, que en los años ochenta se volvió un estándar dada la ausencia de otros modelos.

**Características**

El proceso en cascada se originó como una forma de analizar, diseñar y construir de acuerdo a las necesidades de los usuarios. Esta aproximación en el ciclo de vida del desarrollo de sistemas, describe un método de desarrollo que es lineal y secuencial.

Posee diferentes objetivos en cada fase de desarrollo. Una vez que una fase se completa, se continúa a la siguiente y no hay vuelta atrás.

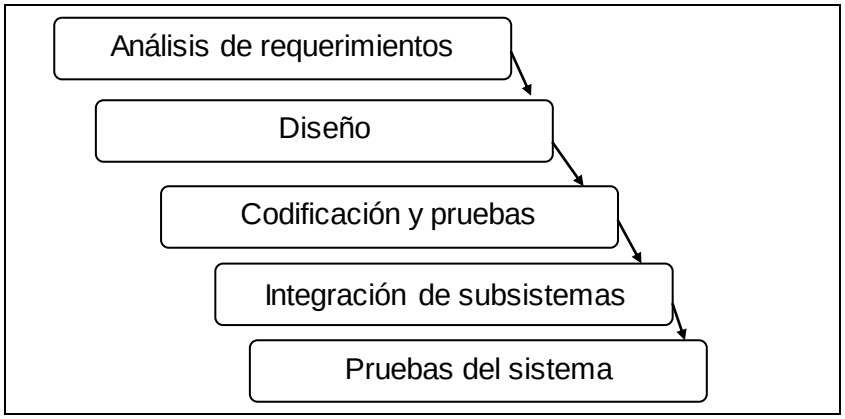



**figura 1- Proceso cascada.**

**Ventajas**

La ventaja de éste modelo es que permite la división en departamentos y el control a nivel de proyecto. Permite fijar fechas límite en la planificación para cada etapa de desarrollo y el producto puede ser llevado de etapa en etapa y llevar a cabo el producto en tiempo.

**Limitaciones**

No permite la revisión. Es decir, una vez que la aplicación está en la fase de prueba, es muy difícil volver hacia atrás y cambiar algo que no fue bien pensado en la fase de análisis de requerimientos.

Retrasa el manejo de riesgos críticos para el proyecto.

Retrasa el *testing* y la integración del sistema.

No promueve el desarrollo temprano (4).

### 1.3.1.2 Proceso iterativo

Surge como mejora al proceso en cascada y se usó de forma exitosa por cuatro décadas en varias organizaciones. Las primeras referencias que se encontraron sobre éste modelo datan del año 1968, también se le conoce como circular o evolutivo.

**Características**

Está compuesto por un conjunto de iteraciones. Cada iteración tiene como objetivo entregar versiones del software y se considera un subproyecto que genera productos de software, no solo documentación. En cada iteración se efectúan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema. Al igual que el proceso en cascada, comienza con una fase de requerimientos, seguida de una fase de diseño y otra

de implementación. Luego de esta primera ronda de implementación, se inicia una fase de evaluación con el fin de verificar el éxito o falla del trabajo completado.

**Ventajas**

La ventaja del uso de éste modelo es que los usuarios finales son involucrados en el proceso de desarrollo. En lugar de esperar hasta que la aplicación sea el producto final, si bien luego podría no ser fácil efectuar cambios, los problemas son identificados y resueltos en cada fase de desarrollo. Otros beneficios son:

- ✓ Incrementar la usabilidad y calidad de la aplicación.
- ✓ Descubrimiento temprano de las fallas.
- ✓ Incorporar rápidamente nuevas funcionalidades.
- ✓ Lograr un equipo más motivado.

**Limitaciones**

Puede tornarse un proyecto muy costoso sí: las iteraciones no son lo suficientemente pequeñas para mitigar los riesgos, la posible dificultad para coordinar proyectos de gran envergadura, la tendencia a no documentar el sistema después de que es completado y la dificultad en predecir exactamente qué funcionalidades están dentro de los tiempos y el presupuesto estimado (4).

### 1.3.1.3 Proceso espiral

**Características**

Fue desarrollado en 1986 y combina las funcionalidades de modelos anteriores pero agregando el análisis de riesgos dentro del proyecto de *software*. Dado que se basa en el modelo de prototipos, permite la entrega, desde las fases iniciales, de versiones del producto ya probados, logrando un proceso continuo de pruebas y retroalimentación.

**Pasos**

1) Los requerimientos del sistema son definidos con el mayor detalle posible.

2) Se efectúa el diseño preliminar.

3) Construcción del primer prototipo a partir del diseño preliminar representando las características del producto final.

4) Se desarrolla un segundo prototipo como sigue:

- ✓ Evaluación del prototipo anterior partiendo de fortalezas, debilidades y riesgos.
- ✓ Definición de requerimientos del segundo prototipo.

- ✓ Planificación y diseño del nuevo prototipo.
- ✓ Construcción y Pruebas.

5) Según la opinión del cliente, el proyecto puede ser abortado si el riesgo es demasiado grande. Entre ellos pueden estar involucrados los costos, errores de cálculo en los costos de operación, etc.

6) Se evalúa el prototipo de la misma forma que el anterior. En caso de ser necesario se desarrolla uno nuevo.

7) Se iteran hasta que el cliente considere al prototipo como el producto final luego de los refinamientos ya practicados.

8) Se construye el sistema final basado en el prototipo.

9) Se evalúa y prueba el sistema. Se practican rutinas de mantenimiento para prevenir fallas.

**Ventajas**

El modelo en espiral es indicado para los desarrollos internos de grandes sistemas.

Los beneficios que aporta son:

- ✓ Mecanismos de reducción de riesgos.
- ✓ Soporta el trabajo en iteraciones y refleja prácticas del mundo real.
- ✓ Es una aproximación sistemática.

**Limitaciones**

- ✓ Requiere experiencia en la evaluación y reducción de riesgos.
- ✓ Es complejo, relativamente dificultoso de seguir estrictamente.
- ✓ Aplicable solo en grandes sistemas (4).

### 1.3.2 Metodología RUP

**Características**

RUP es una metodología basada en el modelo espiral y teniendo un proceso iterativo e incremental de forma que el trabajo pueda dividirse en pequeñas partes o proyectos, logrando un equilibrio entre casos de uso y arquitectura. Cada parte puede verse como una iteración de la cual se obtiene un incremento que produce un crecimiento en el producto. Organiza las iteraciones en etapas y fases proponiéndose obtener una estructura sólida y ajustable a las necesidades particulares de cada organización.

**Ventajas**

Teniendo en cuenta que RUP es una metodología:

- ✓ Con aproximación iterativa: Los riesgos son tomados en consideración de forma temprana, los cambios son más manejables, el equipo del proyecto aprende con la experiencia a lo largo del proceso y existe revisión continua de calidad.
- ✓ Con centro en la arquitectura: Gana control intelectual.
- ✓ Orientada a Casos de Uso: Expresada desde la perspectiva de usuarios, escrita en lenguaje natural y por lo tanto de fácil comprensión, alto grado de trazabilidad, provee una forma simple de descomponer los requerimientos en partes, define casos de prueba y posee planificación de iteraciones.

**Limitaciones**

- ✓ En cuanto al tamaño, es extensivo y de difícil comprensión.
- ✓ Alta complejidad y gran amplitud que requiere su personalización para ajustarse a una determinada compañía antes de poder usarse.
- ✓ Requiere grandes esfuerzos iniciales e inversión para comenzar su uso.
- ✓ El manejo ad hoc de requerimientos.
- ✓ Comunicación imprecisa e incomprensible.
- ✓ Inconsistencia en requerimientos, diseño e implementación.
- ✓ Falta de pruebas.
- ✓ Estado de juicio subjetivo.
- ✓ Inhabilidad para manejar riesgos.
- ✓ Cambios sin control.
- ✓ Automatización de desarrollo ineficiente (4).

### 1.3.3 Metodologías ágiles de desarrollo de software

En esta sección se estudian las metodologías de desarrollo ágiles. Se explicarán sus características en general, haciendo mención de sus ventajas y limitaciones.

#### 1.3.3.1 Manifiesto ágil

El Manifiesto Ágil, es un documento que expresa la filosofía ágil y su distinción de los métodos tradicionales. Según el Manifiesto Ágil se valora lo siguiente:

- ✓ Los individuos y sus interacciones son más importantes que procesos y herramientas.
- ✓ Lo importante es satisfacer al cliente a través de la entrega temprana y continua de *software* con valor.

- ✓ El *software* que funcione es más importante que la documentación exhaustiva.
- ✓ Colaboración con el cliente en lugar de negociación de contratos.
- ✓ Repuesta ante el cambio en vez de seguir un plan cerrado.

**Características**

Lo que hace que un método de desarrollo sea ágil es la simplicidad y la velocidad. En el trabajo de desarrollo, el grupo de desarrollo solo se concentra, en primera instancia, en las funcionalidades básicas del sistema, entregando versiones de forma rápida. Por lo tanto una metodología de desarrollo ágil se puede describir como un desarrollo de *software* incremental (pequeñas versiones con ciclos rápidos), cooperativa (clientes y desarrolladores trabajando constantemente juntos), sencillo (método fácil de aprender, modificar y documentar) y adaptativa (capaz de impactar cambios de último momento).

Por su parte, Jim Highsmith describe las tres principales características de una metodología ágil:

- ✓ Manejo adaptativo: Crear un ambiente con variedad de requisitos para enfrentar el desafío de proyectos extremos y particularmente los de grandes cambios.
- ✓ Valores y principios colaborativos: Muchas metodologías son diseñadas para estandarizar las personas a los proyectos, mientras que las ágiles lo son para capitalizar las habilidades individuales y en equipo y de esta forma, adaptar los procesos a las personas.
- ✓ Apenas lo suficiente: Intenta resolver la pregunta de cuánta estructura es suficiente. La idea es lograr un balance entre flexibilidad y estructura de forma tal que se reduzcan los costos y sobre todo, hacer hincapié en que la creatividad y la innovación ocurren en ambientes no estructurados.

**Ventajas**

- ✓ El desarrollo de software con metodologías ágiles permite la producción de software en pequeñas iteraciones. Minimiza la planificación en etapas iniciales. Al tener una planificación en cada etapa de desarrollo se tiene una metodología más fluida que garantiza mejores resultados.
- ✓ Al predecir solo con unas pocas semanas de anticipación, se minimiza la pérdida de tiempo que lleva la documentación. Se tienen en cuenta las variables de todos los días que pueden afectar cualquier proyecto, por ejemplo cambios de personal,

desarrollos tecnológicos y cambios en regulaciones como son las del sector financiero.

- ✓ Debido a las constantes integraciones y pruebas en cada etapa, el producto final puede versionarse lo antes posible, no bien está listo y los cambios pueden realizarse de forma tal que el desarrollo lleve a una implementación más rápida y exitosa.
- ✓ Las pruebas ejecutadas en cada iteración aseguran que las fallas puedan ser corregidas regularmente. Por otra parte, al desarrollar las pruebas de esta forma se establece una relación colaborativa y más cercana con el usuario quién ayudará a detectarlas fallas y conocerá las correcciones que incluirá la siguiente iteración.

**Limitaciones**

- ✓ Soporte limitado para ambientes de desarrollo distribuido: Los procesos de desarrollo ágiles recomiendan reuniones cara a cara de forma periódica. Cuando el equipo de trabajo y los clientes están distribuidos físicamente esto se vuelve dificultoso.
- ✓ Soporte limitado en subcontratos: Cuando se trata proyectos donde se solicita a una empresa extranjera un determinado desarrollo, es necesario que esta empresa interesada presente una licitación que incluya un plan con procesos, hitos y entregas, con el suficiente detalle a fin de determinar un costo estimado.
- ✓ Soporte limitado en el desarrollo seguro de software crítico: Los mecanismos de control de calidad soportados por los procesos ágiles, todavía no han probado ser adecuados y suficientes para asegurar a los usuarios que el producto sea seguro.
- ✓ Soporte limitado en el desarrollo de sistemas grandes y complejos: En los sistemas complejos o de gran tamaño existen aspectos de arquitectura críticos que pueden dificultar un cambio, debido al rol crítico que desempeñan en los servicios del sistema (4).

### 1.3.3.2 Metodología (Programación Extrema o XP)

Se seleccionaron buenas prácticas de esta metodología después de un estudio abarcador de las metodologías existentes para así poder desarrollar un mejor procedimiento de pruebas. Las buenas prácticas de la metodología XP son las que más se asocian a las

aplicaciones en tarjetas inteligentes y así se garantiza una mayor calidad ya que se realizan pruebas durante todo el desarrollo de las aplicaciones. Otra de las causas de esta selección es que las compañías productoras del hardware necesitan un determinado software de último momento que cumpla con algunos requisitos específicos pero no tienen claridad ni en el acabado del software y a medida que avanza el proyecto van apareciendo nuevos requisitos y nuevas aplicaciones que incluirle al mismo.

**Características**

- ✓ Desarrollo iterativo e incremental: pequeñas mejoras.
- ✓ Programación en parejas: el código es revisado y discutido mientras se escribe.
- ✓ Corrección de todos los errores: hacer entregas frecuentes para revisar lo que se está produciendo antes de añadir nueva funcionalidad al sistema.
- ✓ Propiedad del código compartida: este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto.
- ✓ Simplicidad en el código: es la mejor manera de que las cosas funcionen.

**Ventajas**

- ✓ Programación organizada
- ✓ Menor taza de errores
- ✓ Satisfacción del programador
- ✓ El cliente tiene el control sobre las prioridades
- ✓ Se hacen pruebas continuas durante el proyecto

**Limitaciones**

- ✓ Contar con la gente equivocada. Se debe contar con personas capacitadas técnicamente y con buen trato social, de forma que mantengan buenas comunicaciones verbales y capaces de compartir con el resto de los integrantes de los equipos de trabajo. El equipo de desarrollo debe ser disciplinado.
- ✓ Tener al cliente equivocado. El cliente es uno de los responsables máximos del éxito o fracaso del proyecto, no necesariamente tiene que ser personal altamente capacitado en negocios o directivo, sin embargo con la responsabilidad del proyecto debe venir la jerarquía de mando, de lo contrario no podrá resolver situaciones que tienen que ver con el ambiente exterior al proyecto (4).

## 1.4 Calidad de software

### 1.4.1 Definición de calidad de software

Existen diversas definiciones de la calidad del *software* enunciadas por varias compañías entre ellas la ISO y la IEEE que proponen normas y estándares para llevar a cabo una correcta práctica que garantice la buena ejecución de los procesos, dentro de las cuales pueden citarse: “La calidad del *software* es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario” (5).

“El conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas” (5).

En resumen a todas las definiciones dadas sobre la calidad de *software* se puede decir que: "En concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo ya documentados y con las características implícitas que se espera de todo *software* desarrollado profesionalmente" (5).

Para obtener un *software* con calidad se requiere de la utilización de metodologías y procedimientos estándares para el desarrollo de los requerimientos, el análisis, el diseño, la implementación y finalmente las pruebas del *software*, que son el elemento fundamental para el logro de la calidad de cualquier sistema o parte de este. Las pruebas permiten nivelar la estrategia de trabajo en aras de lograr una mayor confiabilidad, sustentabilidad y facilidad de las soluciones (5).

### 1.4.2 Garantía de la calidad de software

La garantía de la calidad del *software* como algunos autores acostumbran denominarle se define como una actividad que posibilita asegurar y proteger todo el proceso de ingeniería de *software* (5).

Es un conjunto de procedimientos, técnicas y herramientas aplicadas por profesionales durante el ciclo de desarrollo de un producto para asegurar que el producto satisface o excede los estándares o niveles de calidad preestablecidos (5).

Es la guía de los preceptos, de gestión y de las disciplinas de diseño para el espacio tecnológico y la aplicación de la ingeniería del *software* (5).

La garantía de calidad consiste en la auditoría y las funciones de información de la gestión. El objetivo de la garantía de calidad es proporcionar la gestión para informar los datos necesarios sobre la calidad del producto, por lo que se va adquiriendo una visión más profunda y segura de que la calidad del producto está cumpliendo sus objetivos. Por supuesto, si los datos proporcionados mediante la garantía de calidad identifican problemas, es responsabilidad de la gestión afrontar los problemas y aplicar los recursos necesarios para resolver los aspectos de calidad (5).

## 1.5 Pruebas de software

Existen numerosos estilos de presentar las pruebas de *software* uno de ellos es a lo largo de todo el ciclo de vida del *software*, pasando por requerimientos, análisis y diseño, programación, puesta en marcha y mantenimiento. La prueba es un elemento crítico para la calidad del *software*. La importancia de los costos asociados a los errores promueve la definición y aplicación de un proceso de pruebas minuciosas y bien planificadas. Las pruebas permiten validar y verificar el *software*, entendiendo como validación del *software* el proceso que determina si el *software* satisface los requisitos y como verificación el proceso que determina si los productos de una fase satisfacen las condiciones de dicha fase.

### 1.5.1 Definiciones importantes

**Pruebas:** Una actividad en la cual un sistema o uno de sus componentes se ejecutan en circunstancias previamente especificadas, los resultados son observados, registrados y se realiza una evaluación de algún aspecto.

**Caso de prueba:** Un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular.

**Defecto:** Un defecto en el *software* como; un proceso, una definición de datos o un paso de procesamientos incorrectos en un programa.

**Fallo:** La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados.

### 1.5.2 Técnicas de pruebas

**Caja blanca:** Se centra en la estructura interna del programa (analiza los caminos de ejecución).

**Caja negra:** Se centra en las funciones, entradas y salidas.

### 1.5.3 Niveles de pruebas

**De Unidad:** En la Metodología XP son enfocadas al código fuente de los componentes, se utilizan para verificar todos los flujos de control y primero pasa por la revisión del programador. La práctica que se utiliza para desarrollar las pruebas unitarias es Desarrollo Dirigido por Pruebas (TDD): es una técnica de programación que plantea escribir primero los casos de prueba y luego implementar lo necesario para ejecutarla. Antes de escribir cualquier fragmento de código, se debe escribir las pruebas automatizadas para comprobar la funcionalidad de ese futuro código. Como el código no existe, inicialmente la prueba falla. Una vez comenzadas las pruebas, se debe eliminar el código duplicado (6).

**De Integración:** Se comprueba la compatibilidad y funcionalidad de las interfaces entre las distintas partes que componen un sistema, estas partes pueden ser módulos, aplicaciones individuales, aplicaciones cliente/servidor, entre otras. Este tipo de pruebas es especialmente relevante en aplicaciones distribuidas.

Existen dos estrategias de integración:

- ✓ Integración descendente
- ✓ Integración ascendente (6)

Integración descendente: La prueba de integración descendente es un planteamiento incremental a la construcción de la estructura de programas. Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando por el módulo de control principal (programa principal).

- ✓ Se usa el módulo de control principal como controlador de la prueba, disponiendo de resguardos para todos los módulos directamente subordinados al módulo de control principal.
- ✓ Dependiendo del enfoque de integración elegido (primero-en-profundidad o primero-en-anchura) se van sustituyendo los resguardos subordinados uno a uno por los módulos reales.

- ✓ Se llevan a cabo pruebas cada vez que se integra un nuevo módulo.
- ✓ Tras terminar cada conjunto de pruebas, se reemplaza otro resguardo con el módulo real.
- ✓ Se hace la prueba de regresión para asegurarse de que no se han introducido errores nuevos.

El programa continúa desde el paso 2 hasta que se haya construido la estructura del programa entero. Para llevar a cabo las pruebas de integración descendentes es necesario crear resguardos, los cuales son una serie de programas que reemplazan los módulos de bajo nivel. Esto se hace necesario cuando se requiere un proceso de los niveles más bajos de la jerarquía para poder probar adecuadamente los niveles superiores.

Para darle solución a este problema se tienen tres opciones:

- ✓ Retrasar muchas de las pruebas hasta que los resguardos sean reemplazados por los módulos reales.
- ✓ Desarrollar resguardos que realicen funciones limitadas que simulen los módulos reales.
- ✓ Integrar el software desde el fondo de la jerarquía hacia arriba (6).

Integración ascendente**:** La prueba de integración ascendente, empieza la construcción y la prueba con los módulos atómicos. Dado que estos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardos.

Pasos para implementar una estrategia de integración ascendente:

- ✓ Se combinan los módulos de bajo nivel en grupos que realicen una subfunción específica del *software*.
- ✓ Se escribe un controlador para coordinar la entrada y la salida de los casos de prueba.
- ✓ Se prueba el grupo.
- ✓ Se eliminan los controladores y se combinan los grupos moviéndose hacia arriba por la estructura del programa (6).

**De Sistema:** El *software* ya validado se integra con el resto del sistema donde algunos tipos de pruebas a considerar son:

- ✓ Rendimiento: determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones, etc.
- ✓ Resistencia: determinan hasta donde puede soportar el programa determinadas condiciones extremas.
- ✓ Robustez: determinan la capacidad del programa para soportar entradas incorrectas.
- ✓ Seguridad: se determinan los niveles de permiso de usuarios, las operaciones de acceso al sistema y acceso a datos.
- ✓ Usabilidad: se determina la calidad de la experiencia de un usuario en la forma en la que este interactúa con el sistema, se considera la facilidad de uso y el grado de satisfacción del usuario.
- ✓ Instalación: se determinan las operaciones de arranque y actualización del *software* (6).

**De Aceptación:** Son las que hace el cliente para determinar si el sistema cumple con lo deseado por él y se obtiene su conformidad (6).

### 1.5.4 Pruebas

**Pruebas de regresión:** Son las actividades que ayudan a asegurar que los cambios no introduzcan un comportamiento no deseado o errores adicionales. Estas constituyen una estrategia importante para reducir efectos colaterales. Se deben ejecutar pruebas de regresión cada vez que se realice un cambio importante en el *software*.

**Pruebas de humo:** Estas se caracterizan por una estrategia de integración continua. El *software* es reconfigurado con la incorporación de nuevos componentes y utilizado continuamente.

**Pruebas de validación:** Son las pruebas realizadas sobre un *software* completamente integrado para evaluar el cumplimiento con los requisitos especificados.

**Prueba alfa:** Se utiliza el *software* de forma natural con el desarrollador como observador del usuario, registrando los errores y los problemas de uso. La prueba alfa se realiza en un entorno controlado.

**Prueba beta:** A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del *software* en un entorno

que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.

**Prueba de recuperación:** Es una prueba del sistema que fuerza el fallo del *software* de muchas formas y verifica que la recuperación se lleva acabo apropiadamente.

**Prueba de seguridad:** Intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán de accesos impropios.

**Pruebas de resistencia:** Están diseñadas para enfrentar a los programas con situaciones anormales.

**Pruebas de rendimiento:** Están diseñadas para probar el rendimiento del *software* en tiempo de ejecución dentro del contexto de un sistema integrado (6).

## 1.6 Ejemplos de aplicaciones de tarjetas inteligentes

### 1.6.1 Tarjetas telefónicas prepago

Estas ofrecen una solución adecuada para comunicaciones internacionales en cualquier país del mundo, una solución para hacer llamadas telefónicas internacionales. Estas tarjetas telefónicas presentan soluciones efectivas de telefonía prepago hacia cualquier destino mundial, implementando un sistema de alta calidad, con gran beneficio que aportan una gran cantidad de minutos. También presentan la comodidad de enviar el pin de la tarjeta o tarjetas que compre un usuario determinado a su propio correo electrónico, con tarifas telefónicas más reducidas. Las tarjetas telefónicas prepago es actualmente un sistema de alto rendimiento y produce un gran ahorro en los usuarios.

### 1.6.2 Tarjetas SIM de móviles

La tarjeta SIM es un chip a través del cual se identifican las personas ante el operador GSM y a través de este con otros operadores. La tarjeta SIM, consta de un código de seguridad de acceso llamado PIN, el cual está formado por cuatro dígitos. Dicho código se puede cambiar a voluntad del usuario e incluso anular su petición. Presenta características especiales como:

- ✓ Mecanismos de seguridad de hardware en el componente.
- ✓ Protección de integridad de los datos mediante bytes de redundancia lineal.
- ✓ Zona de espejo de seguridad ante extracción indebida de la tarjeta.

### 1.6.3 Tarjeta bancaria

La tarjeta bancaria no es más que un plástico que dotado de una identidad única permite a su titular movilizar fondos desde su cuenta bancaria o crédito establecido hacia sí misma, retirada de efectivo en un cajero, o hacia un tercero, pago en un determinado establecimiento. En el fondo la tarjeta utilizada como medio de pago no es más que una forma de automatizar una transferencia desde la cuenta del cliente hacia la cuenta del proveedor, asumiendo este último el costo de la transferencia y siendo el banco del titular de la tarjeta el que asume los posibles riesgos de insolvencia. Para que esta transferencia de fondos sea posible las tarjetas se emiten en función de una serie de estándares que conforman las distintas redes, Tarjeta 6000, Visa, *Master Card*, *American Express*, *Servired* entre otras. La tarjeta incorpora un chip con información sobre el titular y características de la tarjeta (límite disponible, etc.). Si bien en el ejemplo el chip es visible hoy en día existen tarjetas que incorporan dicho chip pero incrustado en el plástico. Son utilizadas en el transporte público de muchas ciudades, donde tan solo acercando la tarjeta a un lector este reconoce el saldo de la misma y le descuenta el importe del trayecto que va a realizar el usuario. También utiliza un PIN que se trata del número de identificación de la tarjeta, de tal modo que cada plástico emitido es único, con independencia de que sobre una misma cuenta de tarjetas se expidan varios plásticos o tarjetas físicas a favor de distintas personas.

### 1.6.4 Monedero electrónico

El monedero electrónico consiste en una tarjeta que incorpora un pequeño chip en el que se almacena valor monetario pre-pagado, que puede ser gastado en cualquier comercio que haya instalado un lector de tales tarjetas. Las tarjetas telefónicas constituyen un buen ejemplo de monedero, aunque limitado a una única función. Idealmente, los monederos electrónicos serán capaces de utilizarse para pagar cualquier compra.

### 1.6.5 Tarjeta sanitaria

Las tarjetas sanitarias incorporan una serie de datos básicos comunes y estarán vinculadas a un código único de identificación personal para cada ciudadano en el sistema nacional de salud de España. La tarjeta sanitaria individual contendrá, de manera normalizada y de forma visible, los siguientes datos: administración sanitaria emisora de la tarjeta; apellidos y nombre del titular de la tarjeta; código de identificación personal

asignado por la administración sanitaria que emite la tarjeta; modalidad de la prestación farmacéutica y leyenda que informa de su validez en todo el sistema nacional de salud de España.

## 1.7 Ejemplos de procedimientos de pruebas de software

### 1.7.1 Automatización y gestión de las pruebas funcionales usando herramientas de código abierto

La automatización de las pruebas funcionales reduce significativamente el esfuerzo dedicado a las pruebas de regresión en productos que se encuentran en continuo mantenimiento. La automatización de las pruebas debe ser considerada un proyecto en sí mismo con objetivos definidos. Hay herramientas que apoyan diversos aspectos de la prueba. A continuación se presenta una clasificación de estas:

**Administración de las pruebas y el proceso de pruebas:** Herramientas para la administración de las pruebas, para el seguimiento de incidentes, para la gestión de la configuración y para la administración de requerimientos.

**Pruebas estáticas:** Herramientas para apoyar el proceso de revisión, herramientas para el análisis estático y herramientas de modelado.

**Especificación de las pruebas:** Herramientas para el diseño de las pruebas y para la preparación de datos de prueba.

**Ejecución de las pruebas:** Herramientas de ejecución de casos de prueba, herramientas de pruebas unitarias, comparadores, herramientas de medición del cubrimiento, herramientas de seguridad.

**Desempeño y monitorización:** Herramientas de análisis dinámico, herramientas de desempeño, de carga y de estrés, herramientas de monitorización.

Existe un proceso definido para las pruebas funcionales manuales en el Centro de Ensayos de *Software* (7). Dicho proceso cuenta con etapas, actividades, roles y artefactos definidos para un proyecto de prueba independiente, donde el equipo de pruebas es contratado para realizar las pruebas de un producto de *software* desarrollado por terceros. El proceso utilizado se llama *Pro Test*.

**Actividades del proceso:**

- ✓ **AT1 – Definición de las pruebas automatizadas:** El objetivo de esta actividad es definir el conjunto de ciclos funcionales o funcionalidades que se probarán con las pruebas automatizadas. Esta actividad consiste en definir con el Cliente qué pruebas se van a automatizar (en alto nivel), considerando los ciclos funcionales o funcionalidades que involucran. Para ello se priorizan y evalúan las pruebas y las funcionalidades a probar.
- ✓ **AT2 – Definición de los Procedimientos de Prueba:** El objetivo de esta actividad es definir las *suites* y *scripts* que conformarán las pruebas automatizadas. A partir de los ciclos funcionales y funcionalidades seleccionados para las pruebas, se especifican las *suites* y *scripts* que las ejecutarán. Definir las *suites* implica definir los scripts que las componen y especificar posibles dependencias de ejecución entre ellas. Para cada *script* se debe definir la tarea que debe realizar y las verificaciones que debe contener.
- ✓ **AT3 – Generación de *suites* y *scripts*:** El objetivo de esta actividad es obtener las *suites* con sus *scripts* correspondientes. Esta actividad consiste en el armado de cada una de las *suites* correspondientes al ciclo funcional a automatizar. Se graba o codifica las pruebas obteniendo como resultado los *scripts* integrantes de las *suites*. Se verifica el correcto funcionamiento de cada *script*.
- ✓ **AT4 – Ejecución de las pruebas automatizadas del ciclo funcional:** El objetivo de esta actividad es ejecutar una prueba completa de las *suites* correspondientes al ciclo funcional y verificar su correcto funcionamiento. Esta actividad consiste en realizar la prueba completa del ciclo funcional en el entorno preparado para dicho fin. En caso de un funcionamiento incorrecto deben realizarse los ajustes necesarios. En esta actividad se verifica el comportamiento de las *suites* en su conjunto.
- ✓ **AT5 – Investigación y modificación de herramientas:** El objetivo de esta actividad es encontrar soluciones a las necesidades que no pueden satisfacerse con las herramientas de automatización que se manejan. Esta actividad incluye recorrer foros, referencias y buscar antecedentes similares a la necesidad planteada. Luego se analizan las posibles soluciones. Estas incluyen instalar nuevas versiones de las herramientas, modificar las herramientas o extenderlas.

Esta actividad también incluye búsqueda de nuevas herramientas que asistan en la automatización.

- ✓ **AT6 – Configuración del entorno:** El objetivo de esta actividad es configurar el entorno de la aplicación que se desea probar para poder ejecutar las *suites* correctamente y documentar esta configuración. En esta actividad se debe configurar los datos que mantiene la aplicación de manera que permitan la correcta ejecución de las *suites*. Es importante documentar con el detalle suficiente esta configuración.

- ✓ **AT7- Validación de las pruebas automatizadas:** El objetivo de esta actividad es verificar el correcto comportamiento de los scripts, en el ambiente de prueba del cliente y preparar las suites y scripts generados para la validación del cliente. En esta actividad se prueba y eventualmente se ajustan los scripts para el correcto funcionamiento en el ambiente de pruebas del cliente. El cliente valida comparando las pruebas que espera automatizar con las pruebas que los scripts realizan.
- ✓ **AT8 – Organización de las pruebas automatizadas:** El objetivo de esta actividad es gestionar los artefactos generados en el proyecto de automatización. Esta actividad consiste en definir, actualizar y ejecutar los procedimientos para la gestión de los documentos, suites y scripts que se generen en el proyecto de automatización (7).

| ACTIVIDAD | ROLES | ENTRADA | SALIDA |
|---|---|---|---|
| AT1 - Definición de las pruebas automatizadas | Líder Diseñador de Pruebas Cliente | Requerimientos Acta reunión cliente | Documento que especifica las funcionalidades y ciclos funcionales a probar con las pruebas automatizadas |
| AT2 - Definición de los procedimientos de prueba | Diseñador de Pruebas Cliente | Requerimientos Acta reunión cliente Documento obtenido en AT1 | Documento con *Suites* y *Scripts* que componen las pruebas |
| AT3 - Generación de *suites* y *scripts* | Probador | Documento obtenido en AT2 y documento obtenido en AT6 | *Suites* y *Scripts* que las componen |
| AT4 - Ejecución de las pruebas automatizadas del ciclo funcional | Probador | *Suites* y *Scripts* obtenidos en AT3 | *Suites* y *Scripts* verificados |
| AT5 - Investigación y modificación de herramientas | Probador | Herramienta a investigar | Nuevas herramientas (adquiridas o modificadas) y documentación pertinente |
| AT6- Configuración del entorno | Probador | Aplicaciones a probar y documento obtenido en AT2 | Aplicaciones aptas para ser probadas y documentación pertinente |
| AT7 - Validación de las pruebas automatizadas | Cliente Probador | *Suites* y *Scripts* obtenidos en AT4 | *Suites* y *Scripts* verificados y validados |
| AT8 - Organización de las pruebas automatizadas | Diseñador de Pruebas Probador | Artefactos generados en el proyecto | Artefactos gestionados |

**Tabla 1- Actividades, Roles y Artefactos de Entrada y de Salida.**

### 1.7.2 Procedimiento para pruebas de intrusión en aplicaciones Web

La realización de las pruebas se separó en dos niveles siguiendo la filosofía de la compañía *Above Security*. El primer nivel consiste en la evaluación de las vulnerabilidades de la aplicación, donde se recopilan las debilidades encontradas mediante la utilización de herramientas automatizadas y poca intervención manual.

El segundo nivel consiste en la recopilación de evidencias objetivas que demuestren la explotación de las vulnerabilidades detectadas en el primer nivel.

**Las categorías de prueba del procedimiento son:**

- ✓ **Recopilación de Información:** El objetivo de esta categoría es comprobar si la aplicación brinda datos sensibles utilizables por cualquier atacante. Los tipos de prueba incluyen comprobación de firma digital, descubrimiento de aplicaciones, análisis de códigos de error y pruebas de SSL/TLS.
- ✓ **Comprobación de las reglas del Negocio (CRN):** Las pruebas de estas categorías son la comprobación de las reglas del negocio definidas para la aplicación.
- ✓ **Comprobación de la autenticación (CA):** Esta categoría pone a prueba el sistema de autenticación de la aplicación mediante prueba de fuerza bruta y pruebas al recordatorio de contraseñas del sistema.
- ✓ **Validación de datos (VD):** Verifica que todas las entradas de datos estén validadas realizando pruebas de inyección SQL, ORM, LDAP, XML, SSI, procedimientos almacenados y código.

El procedimiento cuenta con cuatro etapas, la primera etapa se basa en la planificación de las pruebas, la segunda en el diseño de las pruebas, la tercera en la ejecución de las pruebas y la cuarta en la documentación y el informe de los resultados obtenidos (8).

### 1.7.3 Una experiencia novedosa para el *testing* desarrollada por un departamento de pruebas de *software*

Este departamento pertenece a CALISOFT centro ubicado en la Universidad de las Ciencias Informáticas. El departamento cuenta con dos grupos fundamentales que son: Grupo de Ingeniería de Pruebas de *Software* (GIPS) y Laboratorio Industrial de Pruebas de *Software* (LIPS).

El GIPS tiene la responsabilidad de diseñar las pruebas, logrando que estos diseños sean bien específicos para que los probadores del LIPS, detecten la mayor cantidad de No Conformidades (NC) posible.

Los tipos de proyecto que se han tenido en cuenta para definir por cada uno, los tipos de prueba a realizar son: Aplicaciones Web, Aplicaciones de Escritorio, Multimedia, Juegos,

Sistemas Operativos. Dentro de estos grandes grupos existen variedades de artefactos, pero de forma general las pruebas a realizar son las mismas, lo que es diferente es el diseño de prueba que debe realizarse en cada caso. A continuación se precisa por cada uno de estos grupos, los tipos de prueba que se realizan hoy en el Departamento de Pruebas de *Software* (DPSW):

| **Tipos de prueba** | **App Web** | **App escritorio** | **Multimedia** | **Juegos** | **Sistemas operativos** |
|---|---|---|---|---|---|
| Pruebas Funcionales | **x** | **x** | **x** | **x** | **x** |
| Pruebas de Seguridad | **x** | **x** | | **x** | **x** |
| Pruebas de Instalación | **x** | **x** | **x** | **x** | **x** |
| Pruebas de Configuración | **x** | **x** | **x** | **x** | **x** |
| Pruebas de Recuperación y Tolerancia a Fallos | **x** | **x** | **x** | **x** | **x** |
| Pruebas de Usabilidad | **x** | **x** | **x** | **x** | **x** |
| Pruebas de Carga | **x** | | | **x** | **x** |
| Pruebas de Estrés | **x** | | | **x** | **x** |

**Tabla 2 – Pruebas que se realizan en el DPSW.**

Además de estos tipos de pruebas, se revisa de forma general en los proyectos la documentación generada durante el desarrollo y que forma parte de los entregables comprometidos por el cliente.

**Las actividades que se realizan en el LIPS para ejecutar las pruebas de liberación son las siguientes:**

- ✓ **Recibir solicitud:** Se recibe la solicitud en el DPSW por parte del cliente.
- ✓ **Definir tipos de prueba:** Atendiendo al tipo de artefacto que se solicite para liberar, serán planificadas las pruebas correspondientes. Esta actividad será ejecutada por el jefe de DPSW. Una vez definido los tipos de pruebas, la solicitud será enviada a los ingenieros de pruebas, responsables de refinar o realizar los diseños, a partir de los tipos de pruebas a ejecutar.
- ✓ **Preparar pruebas:** Los ingenieros de pruebas preparan las pruebas atendiendo al tipo de artefacto.
- ✓ **Reunión de inicio:** Se realiza la reunión de inicio entre el equipo de proyecto y el DPSW, donde se revisa el plan de pruebas, documento que guía todo el proceso.
- ✓ **Pruebas exploratorias:** Se realizan pruebas exploratorias a los artefactos, aplicando los criterios de criticidad.
- ✓ **Ejecutar pruebas:** Ejecutar las iteraciones para la evaluación del artefacto. Estas evaluaciones se realizan utilizando listas de chequeo, casos de prueba y herramientas automatizadas. Durante las iteraciones son detectadas No Conformidades (NC), que se entregan al equipo de desarrollo para su resolución. La gestión de las pruebas se realiza para el registro de las NC y *Subversion* (SVN) para la planificación del trabajo en el LIPS y guardar los Expedientes de Prueba (EPr) de cada uno de los procesos.
- ✓ **Liberar artefacto:** Se libera el artefacto, listo para ser entregado al cliente.
- ✓ **Almacenar resultados:** Almacenar los artefactos generados como parte de las pruebas en el EPr, definido para cada proceso de pruebas en el *Subversion* (9).

## 1.8 Conclusiones parciales

Con el presente capítulo se pudieron conocer las características básicas de las tarjetas inteligentes y los estándares relacionados a las mismas. También se dieron a conocer las

metodologías de software más utilizadas junto con las definiciones de los conceptos de calidad de *software* y garantía de la calidad de *software* así como las definiciones generales de pruebas, los diferentes tipos que existen y su importancia. Finalmente se presentaron algunos ejemplos de procedimientos de pruebas que actualmente se utilizan en Cuba y en el resto del mundo para garantizar la calidad de los productos de *software*. El estudio de estos elementos garantizó poder establecer la estrategia a emplear en el desarrollo del procedimiento de pruebas para aplicaciones en tarjetas inteligentes.

# CAPÍTULO 2: PROCEDIMIENTO PROPUESTO

## 2.1 Introducción

En este capítulo se realizará una investigación de numerosas herramientas y se desarrollará el procedimiento de pruebas que contará con etapas, roles y herramientas para automatizar las pruebas.

## 2.2 Herramientas investigadas

### 2.2.1 *JUnit*

*JUnit* es una herramienta java para la realización de pruebas unitarias de programas, se trata de realizar *test* de pruebas que se ejecutan realizando un registro de los resultados obtenidos. Esta herramienta presenta facilidad de uso en comparación con el trabajo que realiza. Además presenta ventajas como:

- ✓ Creación de conjuntos de pruebas para una clase.
- ✓ Código libre.
- ✓ Bien documentado.
- ✓ Pruebas a diferentes niveles y capas.
- ✓ Extensible (10).

*JUnit* es un conjunto de clases (*framework*) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces *JUnit* devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente del que regresó el método durante la ejecución, *JUnit* devolverá un fallo en el método correspondiente. *JUnit* fue creado por Erich Gamma y Kent Beck y es utilizado en programación para hacer pruebas unitarias de aplicaciones Java. *JUnit* es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación (11).

### 2.2.2 *NUnit*

Hoy en día es un estándar para las pruebas en la plataforma .NET, además simplifica mucho la creación de pruebas y *suites*, así como la ejecución de las mismas. Es una herramienta de código abierto, está basada en *xUnit* y se utiliza para pruebas unitarias. Es desarrollada en C# y sus principales ventajas por la cual la escogen numerosos desarrolladores son:

- ✓ Es multiplataforma, puede ejecutarse en *Windows* y *Linux*.
- ✓ Provee dos formas de ejecutar y administrar las pruebas: una interfaz gráfica y una interfaz de consola.
- ✓ Ofrece una interfaz simple que informa si una prueba o un conjunto de pruebas fallaron o pasaron.
- ✓ Presenta dos tipos de funcionamiento diferentes:

Atributos personalizados: atributos que le indican a *NUnit* cómo interpretar y ejecutar las pruebas implementadas en el método o clase.

Aserciones: son métodos del marco de trabajo de *NUnit* utilizados para comprobar y comparar valores.

*NUnit* está licenciado con *BSD-style*. Tiene varios complementos para ampliar su aplicación, entre los que se encuentran: *NUnit.Forms* para que sea capaz de manejar pruebas de elementos de interfaz de usuario en *WindowsForms*y*NUnit.*ASP, para manejar pruebas de elementos de interfaz de usuario en ASP.NET (12).

### 2.2.3 *JProbe*

JProbe es una solución completa de optimización del rendimiento para entornos Java que permite, en un tiempo mínimo, descubrir, diagnosticar y resolver incidencias de bajo rendimiento en aplicaciones JEE (*Java Enterprise Edition*) y JSE (*Java Standard Edition*) descendiendo su nivel de análisis hasta la línea de código individual. Es una herramienta de análisis de desempeño Java que proporciona una solución integral para ayudar a los desarrolladores a descubrir las causas y corregir los problemas de ejecución y memoria en las aplicaciones Java. *JProbe* permite probar sus aplicaciones sin ningún cambio en el código y se integra fácilmente con servidores de aplicaciones, IDE, JDK (*Java Development Kit*) y sistemas operativos.

*JProbe* ofrece tres tipos de análisis:

- ✓ **Análisis de memoria:** Permite a los desarrolladores identificar y resolver problemas de uso de memoria, para garantizar la eficiencia y la estabilidad de los programas.
- ✓ **Análisis de rendimiento:** Permite a los desarrolladores identificar y resolver cuellos de botella y puntos muertos, para garantizar la óptima ejecución y escalabilidad de los programas.
- ✓ **Cobertura de análisis:** Permite a los desarrolladores identificar líneas de no ejecución de código durante las pruebas de unidad, para garantizar la cobertura de la prueba y corrección del programa (12).

### 2.2.4 Microsoft Visual Studio 2010

**Microsoft Visual Studio 2010** permite realizar distintas pruebas a través de las siguientes herramientas que están integradas a la *suite*: ***Test View***, ***Test List Editor***, ***Test Results***, ***Code Coverage Results***, ***Test Runs*** *y* ***Test Impact View***. Las pruebas que se pueden llevar a cabo con ellas son: pruebas unitarias, pruebas unitarias de base de datos, pruebas de interfaz de usuario (IU) codificadas, pruebas de carga (dentro de la cual están las pruebas de rendimiento web) y pruebas genéricas. La versión de ***Visual Studio*** en la que están presentes todas las funcionalidades de pruebas es la ***Ultimate***.

Las pruebas de integración no están explícitamente soportadas, pero a través de un conjunto de pruebas unitarias, pruebas ordenadas, pruebas web y pruebas genéricas, se puede probar la integración de varios módulos (12).

### 2.2.5 *NetBeans* (seleccionada)

*NetBeans* es un poderoso entorno de desarrollo que permite desarrollar aplicaciones complejas con interacción web, UML, base de datos, aplicaciones para tarjetas inteligentes e inclusive inteligencia artificial. Desde sus inicios fue evolucionando al realizar distintas versiones con muchas mejoras en cada una de ellas hasta que se tomó la decisión de hacer a *NetBeans* código abierto (*open source*).

| Versión | Fecha de lanzamiento |
|---|---|
| *NetBeans* 7.1.2 | Mayo de 2012 |
| *NetBeans* 7.0.1 | 01 de agosto de 2011 |
| *NetBeans* 7.0 | 20 de abril de 2011 |
| *NetBeans* 6.9.1 | 4 de agosto de 2010 |
| *NetBeans* 6.9 | 15 de junio de 2010 |
| *NetBeans* 6.8 | 10 de diciembre de 2009 |
| *NetBeans* 6.7.1 | 27 de julio de 2009 |
| *NetBeans* 6.7 | 29 de junio de 2009 |
| *NetBeans* 6.5 | 25 de noviembre de 2008 |
| *NetBeans* 6.1 | 28 de abril de 2008 |
| *NetBeans* 6.0 | 3 de diciembre de 2007 |
| *NetBeans* 5.5.1 | 24 de mayo de 2007 |
| *NetBeans* 5.5 | 30 de octubre de 2006 |
| *NetBeans* 5.0 | enero de 2006 |
| *NetBeans* 4.1 | mayo de 2005 |
| *NetBeans*4.0 | diciembre de 2004 |
| *NetBeans* 3.6 | abril de 2004 |
| *NetBeans* 3.5 | junio de 2003 |

**Tabla 3 – Evolución del *NetBeans* con sus versiones.**

(13)

Se seleccionó esta herramienta para realizar las pruebas de caja negra a los *applet*, debido a que es una herramienta libre, multiplataforma, con facilidad de uso, asimila el código de javacard y realiza las pruebas con agilidad. Se pueden probar todas las funcionalidades sin dificultades. Además se utiliza como entorno de desarrollo de las pruebas unitarias utilizando componente *javatest* de la herramienta *JTharness*-4.4.

### 2.2.6 *JTharness*-4.4 (seleccionada)

El *JTharness*-4.4 se basa en *javatest* arnés de Oracle, es un propósito general con todas las funciones, flexible y configurable por el instrumento de prueba muy adecuada para la

mayoría de los tipos de prueba de unidad. Originalmente desarrollado como un instrumento de prueba para ejecutar *suites* TCK de prueba, que desde entonces ha evolucionado hasta convertirse en una plataforma de propósito de la prueba general *JTharness*-4.4 está diseñado para configurar y ejecutar conjuntos de pruebas que constan de muchas pruebas discretas e independientes. Es especialmente bueno en la prueba de APIs y los compiladores. Se puede utilizar para ejecutar las pruebas en todas las plataformas Java, desde la plataforma *JavaCard*, para la plataforma Java, *Enterprise Edition* ("Java EE"). Le permite crear conjuntos de pruebas que son autónomos los productos que los clientes pueden fácilmente configurar y ejecutar (14).

## 2.3 Procedimiento Propuesto

### 2.3.1 Nombre

Procedimiento de pruebas para aplicaciones en tarjetas inteligentes.

### 2.3.2 Objetivo

Evaluar las aplicaciones del Departamento de Tarjetas Inteligentes perteneciente al Centro de Identificación y Seguridad Digital. Especificar los roles que intervendrán, las actividades que desarrollarán y los artefactos que se generarán. Orientar sobre las indicaciones y características generales a tener en cuenta para desarrollar las técnicas de pruebas que forman parte del procedimiento, respetando las reglas y las normas presentes en el mismo.

### 2.3.3 Alcance

El procedimiento comprende todas las aplicaciones en tarjetas inteligentes que se desarrollen en el Departamento de Tarjetas Inteligentes perteneciente al Centro de Identificación y Seguridad Digital.

### 2.3.4 Definiciones Importantes

**Requisitos:** Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.

**Cliente:** Es aquella persona que pretende utilizar los servicios que brinda el presente procedimiento.

**Reporte de Prueba:** Es el resultado de la prueba archivado en un documento que se utilizará cuando se requiera.

**Riesgo:** Posibilidad de que no funcione la aplicación.

**Artefactos:** Cualquier información creada, producida, cambiada o utilizada por las personas en el desarrollo del procedimiento.

**No Conformidades**: Defectos y/o sugerencias detectadas por las personas que trabajan en el procedimiento.

**Expediente de prueba:** El artefacto llamado expediente de prueba tiene el propósito de archivar todos los documentos generados durante la etapa de pruebas.

### 2.3.5 Como utilizar las herramientas seleccionadas en el procedimiento

#### 2.3.5.1 Pasos para la utilización del *NetBeans* 7.1 en las pruebas de caja negra de los *applets*:

- ✓ Se abre un nuevo proyecto en el *NetBeans* 7.1 y se crea una nueva clase seleccionando el lenguaje de programación *Java Card* 3 *Platform* y el tipo de clase *Classic Applet*.
- ✓ Se importa el *javacard.framework* y se copia el código del *applet* que se desea probar.
- ✓ Se abre el paquete APDU Scripts y se selecciona el archivo con extensión scr que genera la herramienta por defecto.
- ✓ En esta clase se realizan los envíos de comandos APDU y la herramienta se encarga de enviar la respuesta de estos comandos. Esta es su estructura:

APDU Command (C-APDU: este comando es usado por el lector para enviar información hacia un simulador de una tarjeta.)

| Encabezado (obligatorio) | | | | Cuerpo (opcional) | | |
|---|---|---|---|---|---|---|
| CLA | INS | P1 | P2 | Lc | Data Field | Le |

CLA: Clase de instrucción. Indica la clase y la estructura.

INS: Código de instrucción. Especifica la instrucción del comando.

P1, P2: Parámetros de instrucción. Proveen más información sobre la instrucción.
LC: Número de bytes en el Data Field del APDU.
Data Field: Secuencia de bytes con información.
LE: Cantidad máxima de bytes esperados como respuesta.
APDU Response(R-APDU: este comando es usado por el simulador de la tarjeta para responder al comando enviado por el lector.)

| | | | |
|---|---|---|---|
| Cuerpo (opcional) | Código respuesta (obligatorio) | | |
| Data Field | SW1 | SW2 | |

Data Field: Secuencia de bytes con información.
SW1, SW2: Status Word (palabra de estado): Denotan el estado del procesamiento del comando en la tarjeta.

- ✓ Se obtienen los resultados.

**Análisis de los resultados:**

Para que el *applet* desarrollado no presente ninguna no conformidad la respuesta debe ser SW1: 90, SW2: 00, además de que coincida con los resultados esperados que se encuentren en los diseños de casos de pruebas.

### 2.3.5.2 Pasos para la utilización del componente *javatest* de la herramienta *Jtharness* 4.4 integrado al IDE *NetBeans* 7.1 en las pruebas unitarias hacia los *applets*;

- ✓ Se abre un nuevo proyecto en el IDE *NetBeans* 7.1 y se crea una nueva clase seleccionando el lenguaje de programación *Java* y el tipo de clase *Java Class*.
- ✓ Se abre el *Jtharness* 4.4 ejecutando el componente *javatest*.
- ✓ Se analiza el código de las clases que se encuentran dentro del paquete *BigNum*.
- ✓ En la nueva clase creada en el *NetBeans* 7.1 se importa *java.io.PrintWriter* y se copia una modificación adaptada a las funcionalidades escogidas del código de

- ✓ las clases del paquete *BigNum* de la herramienta *Jtharness* 4.4 escogiendo los datos de entrada.
- ✓ Posteriormente se le realiza un *Main* a la clase realizada y se ejecuta.
- ✓ Después de ejecutada se obtienen los resultados.

**Análisis de los resultados:**

Los resultados que se obtienen dan una panorámica perfecta para tener la idea clara de los objetivos que perciben las funcionalidades a desarrollar en un *applet* determinado.

### 2.3.6 Responsabilidades de las personas que laboran en el procedimiento

**Administrador de la Calidad:**

- ✓ Asegurar la calidad en el proceso de desarrollo del *applet*.
- ✓ Asegurar que el *applet* producido se ajuste a las especificaciones y que esté razonablemente libre de errores.
- ✓ Proporcionar una metodología para realizar las pruebas.
- ✓ Evaluar los resultados que se obtienen en las pruebas de calidad.

**Diseñador de pruebas:**

- ✓ Saber utilizar las herramientas *NetBeans* 7.1 y *Jtharness* 4.4 para poder diseñar los casos de pruebas.
- ✓ Diseñar los casos de pruebas.
- ✓ Evaluar y documentar el resultado de las pruebas realizadas a los *applets*.
- ✓ Definir listas de chequeo.

**Probador:**

- ✓ Saber utilizar las herramientas *NetBeans* 7.1 y *Jtharness* 4.4 para poder ejecutar los casos de pruebas diseñados.
- ✓ Ejecutar las pruebas diseñadas.
- ✓ Anotar los resultados obtenidos.

**Programador:**

- ✓ Saber utilizar las herramientas *NetBeans* 7.1 y *Jtharness* 4.4 para poder ejecutar los casos de pruebas de unidad diseñados.
- ✓ Realizar pruebas unitarias y pruebas de caja blanca.

### 2.3.7 Descripción

#### 2.3.7.1 Elaboración de la estrategia de pruebas

En la presente estrategia de pruebas se tuvo en cuenta los siguientes aspectos:

- ✓ Funcionalidades del *applet* a probar
- ✓ Nivel de pruebas
- ✓ Tipos de pruebas
- ✓ Técnicas de pruebas
- ✓ Herramientas a utilizar para la automatización de las pruebas

Se desarrolla un plan de pruebas con el objetivo de establecer un grupo de actividades, para las cuales existe una fecha de realización y un responsable que es el encargado de darle cumplimiento.

Estas actividades cuentan con un grupo de artefactos, en los cuales se detallan todas las informaciones necesarias que son derivadas de las etapas del procedimiento. Archivar estas informaciones es de vital importancia y se almacenan en los expedientes de pruebas.

#### 2.3.7.2 Desarrollo del procedimiento propuesto:

El procedimiento incluye el nivel de pruebas de unidad y el tipo de prueba funcional. En el procedimiento se define la utilización de las técnicas de pruebas de caja blanca y pruebas de caja negra utilizando para su desarrollo y automatización las herramientas *NetBeans* 7.1 y *JT Harness* 4.4.

Se utilizan en el procedimiento las buenas prácticas de la metodología XP como son los casos de:

- ✓ Desarrollo iterativo e incremental: pequeñas mejoras.
- ✓ Programación en parejas: el código es revisado y discutido mientras se escribe.
- ✓ Simplicidad en el código: es la mejor manera en que las cosas funcionen.
- ✓ Poca documentación ya que está orientada al código.

Debido a que se cuenta con un equipo de desarrollo pequeño y la aplicación de estas características garantiza: una programación mejor organizada y una menor taza de errores lo cual conlleva a una mejor calidad de las aplicaciones en tarjetas inteligentes del Departamento de Tarjetas Inteligentes del Centro de Identificación y Seguridad Digital.

### 2.3.7.3 Descripción del procedimiento por etapas:


**figura 2- Etapas del procedimiento**

**Planificación**

- ✓ En la etapa de planificación se define la estrategia de pruebas a seguir la cual incluye los niveles de pruebas, el tipo de pruebas y las técnicas de pruebas a desarrollar, además de las herramientas a utilizar para su automatización.
- ✓ Se planifica el orden de la realización de las pruebas y se elabora el plan de pruebas, teniendo en cuenta cómo controlarlo y monitorearlo mientras se ejecuta.
- ✓ En esta etapa se conoce previamente el número de *applets* a probar y en el transcurso del tiempo que se deben realizar.
- ✓ Como artefacto de salida se encuentra el plan de pruebas y el cronograma de pruebas, su máximo responsable es el Administrador de Calidad.

**Diseño**

- ✓ En esta etapa se configura el ambiente de pruebas que tiene como responsable el Administrador de Calidad, esta configuración tiene el objetivo de preparar el entorno para las pruebas al tener en cuenta los recursos humanos y dispositivos necesarios para la calidad de la elaboración de las mismas.
- ✓ Se realiza el diseño de casos de pruebas y se determinan las listas de chequeo a desarrollar, como responsable Diseñador de Pruebas.
- ✓ Como artefactos de salida, diseño de casos de pruebas y lista de chequeo, su máximo responsable es el Diseñador de Pruebas.

**Ejecución**

- ✓ Se ejecutan las pruebas de caja negra utilizando la herramienta *NetBeans* 7.1 y se anotan los resultados obtenidos, el responsable de estas actividades es el Probador. Tiene como artefacto de salida reporte de las pruebas de caja negra y el registro de no conformidades.
- ✓ Se ejecutan las pruebas de unidad diseñadas en la etapa anterior auxiliándose de las herramientas *NetBeans* 7.1 y *JT Harness* 4.4 y se aplica la técnica del camino básico. Tienen como responsable de estas actividades al Programador. Como artefactos de salida reporte de las pruebas de unidad y registro de no conformidades.
- ✓ Se revisan los artefactos resultantes archivados en el expediente de prueba, se analizan y registran los errores encontrados y se evalúan los resultados de las pruebas realizadas. Responsable de estas actividades Administrador de Calidad.

## 2.4 Conclusiones parciales

En este capítulo se presenta un estudio de varias herramientas para poder seleccionar las más adecuadas para el procedimiento. Se ha descrito la propuesta de un procedimiento de pruebas para a las aplicaciones en tarjetas inteligentes. Se muestra cómo serán realizadas las diferentes actividades que se encuentran en las etapas del procedimiento y los responsables en cada una de ellas. Se espera que los resultados de este trabajo muestren su fiabilidad, partiendo de que no existe una iniciativa precedente a la que se expone en este trabajo para aplicaciones en tarjetas inteligentes en el Departamento de Tarjetas Inteligentes del Centro de Identificación y Seguridad Digital.


# CAPÍTULO 3: VALIDACIÓN DEL PROCEDIMIENTO DE PRUEBAS

## 3.1 Introducción

Una vez concluido el procedimiento propuesto para probar las aplicaciones en tarjetas inteligentes, se procede a validar la eficiencia del mismo. Dicha validación se hará mediante la técnica de evaluación del método de expertos Delphi, puesto que es considerado uno de los métodos subjetivos de pronosticación más confiable. La calidad de los resultados depende, sobre todo, del cuidado que se ponga en la elaboración de la encuesta y en la elección de los expertos consultados. Además se le aplica el procedimiento al *applet* Cartera desarrollado en el Departamento de Tarjetas Inteligentes del Centro de Identificación y Seguridad Digital.

## 3.2 Métodos de Expertos

Esta técnica se basa en la consulta a personas que tienen grandes conocimientos sobre el entorno en el que la organización desarrolla su labor. Estas personas exponen sus ideas y finalmente se redacta un informe en el que se indican cuáles son, en su opinión, las posibles alternativas que se tendrán en el futuro.

Son propicias para realizar esta técnica las siguientes condiciones:

- ✓ No existen datos históricos con los que trabajar. Un caso típico, es la previsión de implantación de nuevas tecnologías.
- ✓ El impacto de los factores externos tiene más influencia en la evolución que el de los internos. Así, la aparición de una legislación favorable y reguladora y el apoyo por parte de algunas empresas a determinadas tecnologías pueden provocar un gran desarrollo de éstas, de otra manera hubiese sido más lento.
- ✓ Las consideraciones éticas o morales dominan sobre las económicas y tecnológicas en un proceso evolutivo. En este caso, una tecnología puede ver dificultado su desarrollo si éste provoca un alto rechazo en la sociedad (un ejemplo

lo tenemos en la tecnología genética, que ve dificultado su avance por los problemas morales que implica la posibilidad de manipulación del genotipo) (15).

Sus principales ventajas son:

- ✓ La información disponible está siempre más contrastada que aquella de la que dispone el participante mejor preparado, es decir, que la del experto más versado en el tema. Esta afirmación se basa en la idea de que varias personas aportan más conocimiento que una sola persona.
- ✓ El número de factores que es considerado por un grupo es mayor que los factores que tendría en cuenta una sola persona. Cada experto podrá aportar a la discusión general la idea que tiene sobre el tema debatido desde su área de conocimiento (15).

### 3.2.1 Proceso de selección de expertos

Se entiende por experto a una persona que tiene conocimiento y experiencia en un campo, un especialista que sea capaz de interpretar correctamente las informaciones sobre dicho campo, de ofrecer valoraciones conclusivas de un problema en cuestión y hacer recomendaciones al respecto. En el desarrollo de este proceso se consideraron tres etapas cruciales:

- ✓ Determinar la cantidad de expertos a seleccionar.
- ✓ Conformar el listado de los expertos.
- ✓ Confirmar la participación de los expertos.

Se tiene en cuenta que ningún experto conoce las identidades y respuestas de los otros que componen el grupo, para lograr así que cada uno defienda sus opiniones y en caso de ser erróneas, no habrá pérdida de su prestigio.

### 3.2.2 Determinar la cantidad de expertos

La definición de las funciones requiere el trabajo en grupo de expertos, por lo que se hizo necesario el cálculo de la cantidad requerida de estos para conformar el panel. Teniendo en cuenta que el objetivo fundamental del trabajo de los expertos estaba relacionado con la inclusión o no de determinadas funciones objetivo, el cual se ajusta a las condiciones establecidas por la distribución binomial, se decidió realizar este cálculo a partir de un método basado en esta distribución.

$$n_e = \frac{p(1-p)k}{i^2}$$

Fórmula – Cantidad de Expertos

Donde:

ne: Cantidad necesaria de expertos.

p: Error estimado.

i: Precisión deseada en la estimación.

k: Constante computarizada que depende del nivel de confianza (1-α)

| (1- α) | K |
|---|---|
| 0.90 | 2.6896 |
| 0.95 | 3.8416 |
| 0.99 | 6.6564 |

El número de expertos (ocho) fue calculado a partir de establecer los valores siguientes: p = 0,01; i = 0,1 y 1-α = 0,99. Se seleccionaron, como miembros del grupo de expertos, a aquellos con conocimientos de la disciplina de calidad y tarjetas inteligentes, así como personal calificado y con experiencia (15).

### 3.2.3 Conformar el listado de expertos

Se tuvo en cuenta una serie de cualidades y criterios definidos a continuación:

- ✓ Seriedad.
- ✓ Honestidad.
- ✓ Sinceridad.
- ✓ Responsabilidad.
- ✓ Creatividad.
- ✓ Capacidad de análisis.
- ✓ Graduado del nivel superior.
- ✓ Vinculación al desarrollo de proyectos productivos.
- ✓ Reconocido prestigio profesional en el conocimiento sobre pruebas de calidad de *software*.
- ✓ Reconocido prestigio profesional en el campo de las tarjetas inteligentes y la programación de las aplicaciones relacionada con las mismas.

Estas cualidades y criterios han permitido que las opiniones brindadas sean confiables y válidas para el objetivo propuesto.

### 3.2.4 Confirmar participación de expertos

Una vez conformado el listado, se invitó personalmente a cada experto elegido para participar en la evaluación. Allí se les explicó en qué consistía el trabajo en general, el procedimiento a evaluar y el objetivo de la realización de la encuesta, así como el plazo de entrega. Una vez recibida la respuesta positiva, se estableció el listado final de los expertos, informando a cada especialista su inclusión en el proceso a evaluar y las

instrucciones necesarias para contestar las preguntas. De esta forma culmina el proceso de selección, logrando la participación de los ocho expertos escogidos.

### 3.3 Elaboración de la encuesta

En la confección de la encuesta se tuvo en cuenta los aspectos principales que debe tener la propuesta para su aplicación efectiva en el proceso productivo de la línea de tarjetas inteligentes en el Centro de Identificación y Seguridad Digital. La encuesta está compuesta por preguntas con enfoque investigativo, brindando la posibilidad de que cada experto dé su opinión de manera anónima y a su vez abierta, proporcionando riqueza en sus respuestas. Se les facilitó la posibilidad de modificar aspectos que ellos consideraban necesarios cambiar y presentar su opinión general, a favor o en contra del procedimiento propuesto, con la libertad de expresar todo lo que se pudo obviar en la encuesta. Se les fue a ver a cada uno personalmente, se le explicó detalladamente a cada experto los objetivos y resultados del procedimiento, luego se les dio la encuesta con un plazo de tiempo para entregarla. La encuesta establece una serie de preguntas que permiten visualizar la posibilidad real de aplicar la propuesta.

### 3.4 Resultados de la encuesta

Los resultados de la encuesta realizada se consideran satisfactorios, el 87.50% de los expertos consultados consideraron que el procedimiento estaba a la altura de las necesidades de un proceso de pruebas para el desarrollo de aplicaciones en tarjetas inteligentes. Algunos recomendaron seguir investigando en esa línea para aumentar los avances científicos del procedimiento y así poder comprobar la calidad de *applets* de mayor magnitud.

### 3.5 Ejemplo del procedimiento:

**Procedimiento aplicado al *applet* cartera:**

- ✓ Se utilizará el nivel de pruebas de unidad, el tipo de prueba funcional, las técnicas de caja blanca utilizando la herramienta *NetBeans 7.1* con el componente *javatest* y las técnicas de caja negra con la herramienta *Netbeans* 7.1 en la automatización de las pruebas.
- ✓ Plan de pruebas:

| Fecha(inicio) | Fecha(fin) | Actividad | Responsable |
|---|---|---|---|
| dd/mm/aaaa | dd/mm/aaaa | Configurar ambiente de pruebas. | Administrador de Calidad |
| dd/mm/aaaa | dd/mm/aaaa | Determinar casos de pruebas y listas de chequeo. | Diseñador de Pruebas |
| dd/mm/aaaa | dd/mm/aaaa | Diseñar casos de pruebas de unidad. | Diseñador de Pruebas |
| dd/mm/aaaa | dd/mm/aaaa | Diseñar casos de pruebas de caja negra. | Diseñador de Pruebas |
| dd/mm/aaaa | dd/mm/aaaa | Realizar cronograma de pruebas. | Administrador de Calidad |
| dd/mm/aaaa | dd/mm/aaaa | Evaluar el cumplimiento del cronograma de pruebas. | Programador, Probador |
| dd/mm/aaaa | dd/mm/aaaa | Evaluar los resultados obtenidos en las pruebas. | Administrador de Calidad |

**Tabla 4 - Plan de pruebas.**

- ✓ Cronograma de pruebas:

| Applet | Fecha | Técnicas de pruebas | Funcionalidad |
|---|---|---|---|
| Cartera | dd/mm/aaaa | Caja blanca | ValidatePIN |
| Cartera | dd/mm/aaaa | Camino básico | ValidatePIN |

| | | | |
|---|---|---|---|
| Cartera | dd/mm/aaaa | Caja negra | ValidatePIN |
| Cartera | dd/mm/aaaa | Caja blanca | Credit |
| Cartera | dd/mm/aaaa | Camino básico | Credit |
| Cartera | dd/mm/aaaa | Caja negra | Credit |
| Cartera | dd/mm/aaaa | Caja blanca | Debit |
| Cartera | dd/mm/aaaa | Camino básico | Debit |
| Cartera | dd/mm/aaaa | Caja negra | Debit |

**Tabla 5 - Cronograma de pruebas.**

- ✓ Diseño de casos de pruebas de unidad:

Descripción General:

El *applet* Cartera se encarga de contabilizar el dinero que se encuentre en una cartera electrónica por lo que presenta funcionalidades de suma y resta siempre devolviendo el valor final de la operación.

Funcionalidades:

| **Nombre** | **Descripción de la funcionalidad** |
|---|---|
| ValidatePin | Comprobar que el pin que entra el usuario coincide con el pin que está almacenado en la tarjeta. |
| Credit | Realiza la suma de dos valores entrados. |
| Debit | Realiza la resta de dos valores entrados. |

**Tabla 6 - Funcionalidades y su descripción.**

ValidatePin:

| Variable 1 | Variable 2 | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|
| 1 2 3 4 | | OK*****OK | OK*****OK |
| -1 2 3 | | -1 2 3!= 1 2 3 4 | -1 2 3!= 1 2 3 4 |

**Tabla 7 - Pruebas de unidad a ValidatePin.**

Credit:

| Variable 1 | Variable 2 | Variable resultado | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|
| 12 | 12 | 25 | 12,12,24 | 12,12,24 |
| -1 | 0 | -1 | OK*****OK | OK*****OK |
| 32767 | 2 | 25 | El valor de la suma excede el rango | El valor de la suma excede el rango |

**Tabla 8 - Pruebas de unidad a Credit.**

Debit:

| Variable 1 | Variable 2 | Variable resultado | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|
| 12 | 12 | 25 | 12,12,0 | 12,12,0 |
| -1 | 0 | -1 | OK*****OK | OK*****OK |
| -47 | 32765 | -32812 | El valor de la resta excede el rango | El valor de la resta excede el rango |

**Tabla 9 - Pruebas de unidad a Debit.**

Reporte de Pruebas de Unidad

| | |
|---|---|
| Pruebas de Unidad | |
| Nombre Prueba: Caja blanca | |
| Estado: Satisfactoria | Fecha de la última ejecución: dd/mm/aaaa |
| Ejecutado por: Programador | Verificado por: Administrador de Calidad |
| Descripción: En esta prueba se modelan las funcionalidades más importantes del *applet* Cartera para tener una idea clara de cómo deben funcionar estas. | |
| Resultado: Satisfactorio, no se presentaron No Conformidades. | |

**Tabla 10 – Reporte de pruebas de unidad.**

- ✓ Diseño de casos de pruebas de caja negra:

Descripción General:

El *applet* Cartera se encarga de contabilizar el dinero que se encuentre en una cartera electrónica por lo que presenta funcionalidades de suma y resta siempre devolviendo el valor final de la operación.

Funcionalidades:

| **Nombre** | **Descripción de la funcionalidad** |
|---|---|

| | |
|---|---|
| ValidatePin | Comprobar que el pin que entra el usuario coincide con el pin que está almacenado en la tarjeta. |
| Credit | Realiza la suma de dos valores entrados. |
| Debit | Realiza la resta de dos valores entrados. |

**Tabla 11 - Funcionalidades y su descripción.**

ValidatePin:

| Variable 1 | Variable 2 | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|
| 1 2 3 4 | | SW1: 90<br>SW2: 00 | SW1: 90<br>SW2: 00 |
| -1 2 3 | | Error lexical "-" | Error lexical "-" |

**Tabla 12 - Pruebas de caja negra a ValidatePin.**

Credit:

| Variable 1 | Variable 2 | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|
| 32 | 0 | 32 | 32 |

**Tabla 13 - Pruebas de caja negra a Credit.**

Debit:

| Variable 1 | Variable 2 | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|
| 32 | 14 | 1e | 1e |

**Tabla 14 - Pruebas de caja negra a Debit.**

Reporte de Pruebas de Caja negra

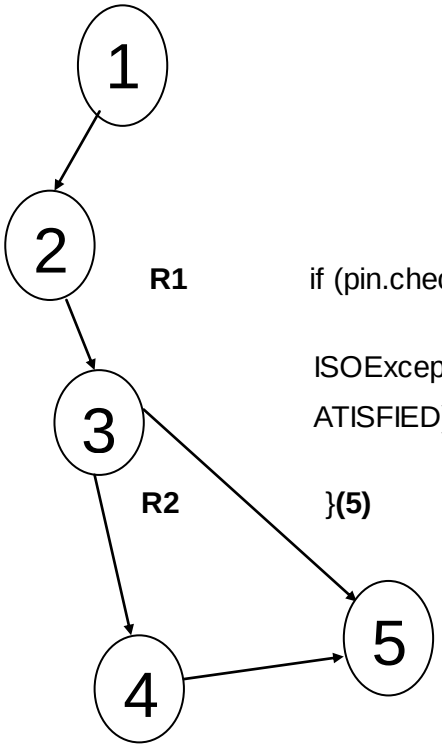
| | |
|---|---|
| Pruebas de Caja negra | |
| Nombre Prueba: Caja negra | |
| Estado: Satisfactoria | Fecha de la última ejecución: dd/mm/aaaa |
| Ejecutado por: Probador | Verificado por: Administrador de Calidad |
| Descripción: En esta prueba se prueban las funcionalidades más importantes del *applet* Cartera. | |
| Resultado: Satisfactorio, no se presentaron No Conformidades. | |

**Tabla 15 – Reporte de pruebas de caja negra.**

- ✓ Técnica del Camino Básico

Funcionalidad ValidatePin:

```
public void ValidatePIN(APDU  apdu, byte[] apduBuffer)
{
byte[] buffer = apdu.getBuffer();(1)
 bytebyteRead = (byte)(apdu.setIncomingAndReceive());(2)
```




```
if (pin.check(buffer, ISO7816.OFFSET_CDATA, byteRead)==false)(3)

ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_S
ATISFIED);(4)

}(5)
```

Complejidad Ciclomática.

V (G): Número de regiones del grafo.

V (G): A – N + 2

V (G): P + 1

A: Número de aristas del grafo. N: Número de nodos.

P: Número de nodos predicados. V (G) = 2

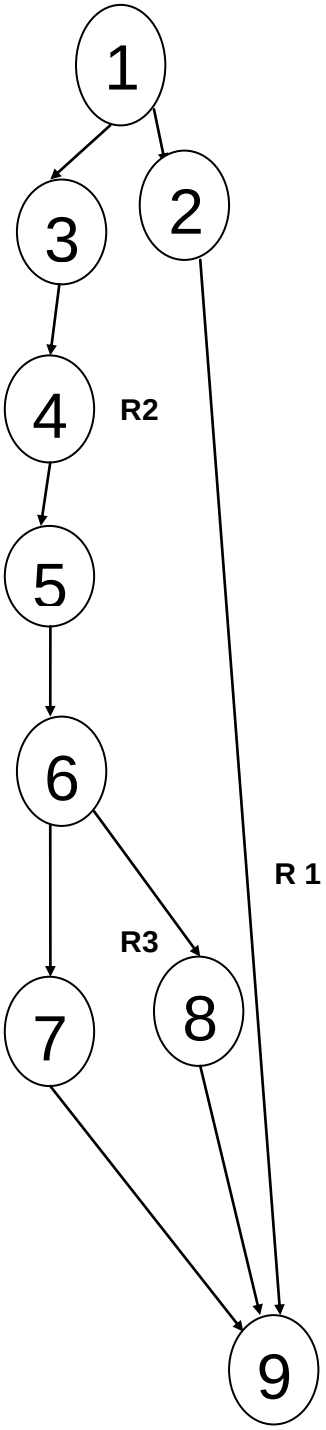
Caminos posibles: 1-2-3-4-5, 1-2-3-5.

R: Región del grafo.

Funcionalidad Credit:

```
private void Credit(APDU apdu, byte[] apduBuffer)
{
    if ( ! pin.isValidated() )(1)
    {

ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
    } (2) else
```




```
{
short amount = GetAmount(apdu, apduBuffer);(3)
short tempBalance = (short)(_Balance + amount);(4)
 if (Validate(tempBalance))(5)
{
 _Balance = tempBalance;(6)
 Response(apdu, apduBuffer);(7)
 }  else
ISOException.throwIt((short)ISO7816.SW_DATA_INVALID);(8)
}
}(9)
```

Complejidad Ciclomática.

V (G): Número de regiones del grafo.

V (G): A – N + 2

V (G): P + 1

V (G) = 3

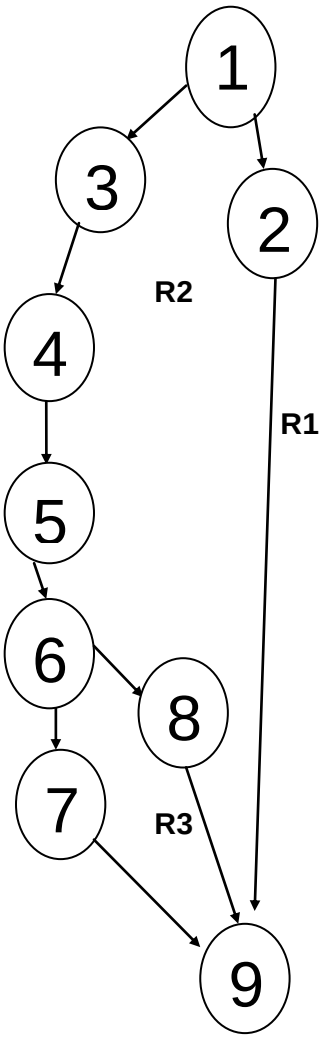
A: Número de aristas del grafo.

N: Número de nodos.

R: Región del grafo.

P: Número de nodos predicados.

Caminos posibles: 1-3-4-5-6-7-9, 1-3-4-5-6-8-9, 1-2-9.




Funcionalidad Debit:

```
private void Credit(APDU apdu, byte[] apduBuffer)
{
 if ( ! pin.isValidated() )(1)
 {
  ISOException.throwIt
  (ISO7816.SW_CONDITIONS_NOT_SATISFIED);
  } (2) else
{
short amount = GetAmount(apdu, apduBuffer);(3)
short tempBalance = (short)(_Balance - amount);(4)
 if (Validate(tempBalance))(5)
{
 _Balance = tempBalance;(6)
 Response(apdu, apduBuffer);(7)
 }  else
ISOException.throwIt((short)ISO7816.SW_DATA_INVALID);(8)
}
}(9)
```

Complejidad Ciclomática.

V (G): Número de regiones del grafo.

V (G): A – N + 2

V (G): P + 1

A: Número de aristas del grafo.

N: Número de nodos.

R: Región del grafo.

P: Número de nodos predicados.

V (G) = 3

Caminos posibles: 1-3-4-5-6-7-9, 1-3-4-5-6-8-9, 1-2-9.

- ✓ Evaluación de los resultados de las pruebas

No se detectó ninguna no conformidad en las principales funcionalidades del *applet* Cartera por lo que se encuentra en perfecto estado y bien estructurado en su código.

## 3.6 Conclusiones Parciales

En este capítulo se logró validar la solución propuesta del procedimiento de pruebas para aplicaciones en tarjetas inteligentes aplicándolo hacia el *applet* Cartera y además utilizando el método de validación de expertos Delphi. Se seleccionaron 8 expertos que respondieron la encuesta y así se lograron obtener los objetivos perseguidos por la propuesta, los resultados obtenidos fueron satisfactorios. Se obtuvo una muestra de los datos obtenidos en las entrevistas a los expertos mediante gráficos. Además se le aplicó el procedimiento al *applet* Cartera.

# CONCLUSIONES

El procedimiento de pruebas para aplicaciones en tarjetas inteligentes introduce mejoras en el proceso de prueba de calidad para los proyectos que utilicen esta tecnología, que permiten revelar las causas de errores, insuficiencias de funcionalidad o discrepancias funcionales con respecto al comportamiento que se esperaba del *applet*. El presente trabajo logra como resultado un modelo flexible y acorde a los requisitos necesarios para la realización de pruebas de calidad en el Centro de Identificación y Seguridad Digital a las aplicaciones en tarjetas inteligentes, además puede ser utilizado por cualquier empresa o proyecto de desarrollo que siga una línea de tarjetas inteligentes. El mismo está compuesto por los entregables que recogerán todos los datos de cada período, los roles involucrados y además se definen herramientas de pruebas. Para la validación de la solución propuesta se le aplicó el procedimiento al *applet* Cartera. También se utilizó el método de expertos Delphi, con la ayuda de un grupo de especialistas, donde el resultado obtenido fue satisfactorio, pudiendo concluir además que este procedimiento permite un control exhaustivo de las fallas que ocurren en las aplicaciones en tarjetas inteligentes facilitando la toma de acciones correctivas para con las mismas.

# RECOMENDACIONES

De acuerdo con la investigación desarrollada, se puede decir que para la mejora del presente procedimiento en un futuro se recomienda:

- ✓ Aumentar los tipos de pruebas a desarrollar en el procedimiento.
- ✓ Continuar desarrollando y documentando el proceso de automatización de las pruebas propuestas.

Para la implantación del procedimiento se recomienda:

- ✓ Impartir asignaturas optativas a los estudiantes del proyecto sobre las pruebas y la calidad de software.
- ✓ Seguir estudiando y profundizando en los conocimientos sobre la calidad de las aplicaciones en tarjetas inteligentes a nivel internacional y como comprobar la misma.
- ✓ Incluirlo en el departamento de tarjetas inteligentes para el desarrollo de sus aplicaciones.

## BIBLIOGRAFÍA

## BIBLIOGRAFÍA

1. **Figueira, Carlos.** Criptografía y Seguridad de Datos. [Online] 2012. http://ldc.usb.ve/~figueira/Cursos/Seguridad/.

2. *Capitulo II - Tecnología de Tarjetas Inteligentes.* **Ramirez, Julio César Castillo.**

3. *Plataforma para el desarrollo de servicios en línea.* **Viñolo, Vismar Fernández Santana y Katerina Pereda.** La Habana : s.n., 2010.

4. *METODOLOGÍAS DE DESARROLLO DE SOFTWARE APLICADAS A SOA.* **Blanco, María Luz.** Buenos Aires República Argentina : s.n., 2007.

5. **EcuRed.** EcuRed. *EcuRed.* [Online] [Cited: octubre 28, 2011.] http://www.ecured.cu/index.php/Pruebas_de_Calidad_de_Software.

6. **Pressman.** *Ingenieria de Software Un Enfoque Practico Pressman 5th Ed.* 2008.

7. **Ignacio Esmite, Mauricio Farías, Nicolás Farías, Beatriz Pérez.** *Automatización y Gestión de las Pruebas Funcionales usando Herramientas Open Source.*

8. *Procedimiento para pruebas de intrusión en aplicaciones.* **Delmys Pozo Zulueta, Mairelis Quintero Ríos, Violena Hernández Aguilar, Lisney Gil Loro, Maria Felix Lorenzo Álvarez.** 2, 2009, Vol. 5.

9. *Una experiencia novedosa para el testing desarrollada por un departamento de pruebas de software.* **Ailyn Febles Estrada, Tayché Capote García, Yeniset León Perdoo, Alionuska Velázquez Cintra, Ramsés Delgado Martínez, Roig Calzadilla Díaz.**

10. *Desarrollo ágil en J2EE con herramientas Open Source.* **Rodriguez, Jose María Alvarez.**

11. *Procedimiento para la realización de pruebas de caja blanca en la UCID.* **Silveira, Andy Abelarde.** La Habana : s.n., 2010.

12. *Propuesta de herramientas para la automatización de las pruebas de software.* **Rachel Hernández Pérez, Yuniel Martínez González.** Habana : s.n., 2011.

13. **Peñarrieta, Ronald.** javaagricola.wikispaces.com. [Online] Mayo 25, 2011. http://javaagricola.wikispaces.com.

14. **Oracle.** download.java.net. [Online] 2011. http://www.download.java.net.

15. *Propuesta de un modelo de pruebas a aplicar durante el ciclo de vida del software de tarjetas inteligentes.* **Jardines, Ander Sánchez.** Habana-Cuba : s.n., 2010.

## BIBLIOGRAFÍA

16. **Tecnología Transaccional.** Tecnología Transaccional. *Tecnología Transaccional.* [Online] [Cited: octubre 25, 2011.] http://tecnologiatransaccional.com/Soluciones/Identificaci%C3%B3nAsistenciaAccesoyBancarias/Productos/Tecnolog%C3%ADa/tabid/73/Default.aspx.

17. **Pressman, R.** *Ingeniería del software:Un Enfoque práctico.* s.l. : McGraw Hill, 2002.

18. *PROCEDIMIENTO GENERAL DE PRUEBAS DE CAJA BLANCA APLICANDO LA TÉCNICA DEL CAMINO BÁSICO.* **Hechavarría, Yaimí Márquez Alpízar y Yenni Valdés.** La Habana : s.n., 2007.

19. *Propuesta de manual de procedimiento de Pruebas de Sistema y su aplicación en el Proyecto CICPC.* **Torres, Yuniet del Carmen Toll Palma y Yilennis Mendoza.** La Habana : s.n., 2007.

20. *Propuesta de procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2.* **Quintana, Marlen Thaureaux Martínez y Dania Rosa Hernández.** La Habana : s.n., 2009.

21. *Procedimiento para pruebas de rendimiento de Carga y Estrés al Sistema Único de Aduanas.* **Figueredo, Anay Caridad Barban Frómeta y Paula Zenaida Hernández.** La Habana : s.n., 2009.

22. *Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS.* **Morales, Dayanis Isaac.** La Habana : s.n., 2009.

23. *Procedimiento para la realización de pruebas de unidad dentro del proyecto Sistema Único de Aduanas.* **Sánchez, Elizabeth Quintas.** La Habana : s.n., 2010.

24. *Procedimiento para pruebas de software con herramientas automatizadas en el Departamento de Pruebas de Software.* **Bravo, Ariannis Marzán Matos y Juan Carlos Villar.** La Habana : s.n., 2010.

25. *PRUEBA DE COMPONENTES DE SOFTWARE BASADAS EN EL MODELO DE JAVABEANS.* **ELIZONDO, PERLA INÉS VELASCO.** TLAXCALA : s.n., 2001.

26. *Implementación del componente réplica de base de datos para Akademos v2.0.* **Pérez, Rosell Pupo Polanco y Yenier González.** La Habana : s.n., 2009.

27. *Una innovadora metodología para el desarrollo de software en ambientes de trabajo virtuales.* **Mitaritonna, Alejandro Daniel.** Buenos Aires : s.n., 2010.

28. *Nuevo Marco de Autenticación para Tarjetas Inteligentes en Red. Aplicación al Pago Electrónico en entornos Inalámbricos.* **Márquez, D. Joaquín Torres.** Madrid : s.n., 2006.

29. *Las Pruebas del Software.* **Tuya, Javier.** Cádiz : s.n., 2007.

30. *TCK Development Guidelines version 1.0.* **Sun Microsystems.** California : s.n., 2001.

# BIBLIOGRAFÍA


31. *ESTUDIO COMPARATIVO DE LOS MODELOS Y ESTANDARES DE CALIDAD DEL SOFTWARE.* **SCALONE, LIC. FERNANDA.** Buenos Aires : s.n., 2006.

32. *Gestión de las Pruebas Funcionales.* **Lamancha, Beatriz Pérez.** Montevideo : s.n., 2007.

33. *Aplicación práctica del diseño de pruebas de software a nivel de programación.* **Cortés, Oscar Hernando Guzmán.** 2004.

34. *Sistema de Apoyo Gerencial Universitario.* **Nader, Ing. Javier.** 2003.

35. *Mejoramiento del Proceso de Pruebas en un Contexto de Desarrollo de Software Globalizado.* **Rubby Casallas, Darío Correal, Nicolás López.** Bogotá : s.n., 2005.

36. *Graphical User Interface.* **Sun Microsystems, Inc.** California : s.n., 2009.

37. *Propuesta de procedimiento general para pruebas de liberación de los productos de la Facultad 1.* **Medina, Noralis Aldana La´O y Jorge Luis Ramos.** La Habana : s.n., 2009.

38. *Propuesta de un procedimiento de pruebas de software en el área temática Sistema de Apoyo a la Salud.* **Quiñones, Elsy Expósito González y Karina Mileisis Torres.** La Habana : s.n., 2008.

39. *Plataforma para el acceso a servicios desde dispositivos móviles utilizando parámetros de autenticación basados en SIM Card.* **Martínez, Francisco, et al., et al.** 26, Bogotá : s.n., 2007.

40. *Procedimiento para pruebas de intrusión en aplicaciones Web.* **Pozo Zulueta, Delmys, et al., et al.** 2, Madrid : s.n., 2009, Vol. 5.

41. *Estudio de la metodología de automatización del testeo en aplicaciones web para e-Catalunya.* **Fariña, Sergio Mendoza.** Barcelona : s.n., 2011.

# GLOSARIO DE TÉRMINOS

**A**

**Artefacto*:*** Pieza de información tangible que es creada, modificada y usada por los trabajadores al realizar las actividades; representa un área de responsabilidad y es candidata a ser tenida en cuenta para el control de la configuración. Un artefacto puede ser un modelo, un elemento de un modelo, o un documento. Véase trabajador, actividad.

**C**

**Calidad:** Calidad de software. Satisfacción de las necesidades de los usuarios.

**Código:** Se refiere a las instrucciones contenidas en un programa, y entendibles por el ordenador. Las aplicaciones normales pueden tener miles de líneas de código que es necesario mantener y actualizar.

**E**

**Estándar:** Es cualquier tecnología aprobada por un comité formalizado.

**H**

**Herramienta:** Subprograma o módulo encargado de funciones específicas y afines entre sí para realizar una tarea. Una aplicación o programa cuenta con múltiples herramientas a su disposición. Por ejemplo, el corrector ortográfico puede ser una herramienta en una aplicación para redactar documentos, pero no es una aplicación en sí misma.

**I**

**Iteración:** Es un ciclo repetitivo de un proceso.

**M**

**Metodología:** Colección de métodos de solución de problemas organizados bajo una filosofía común y gobernada por un conjunto de principios. Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal.

**Modelo:** Representación de la realidad por abstracción.

**P**

**Portabilidad:** Característica de ciertos programas que les permite ser utilizados en distintos ordenadores sin que precisen modificaciones de importancia.

# GLOSARIO DE TÉRMINOS

**Procedimiento:** Dentro de una aplicación, se denomina procedimiento al conjunto de instrucciones, controles, etc. que hacen posible la resolución de una cuestión específica. La impresión es un procedimiento, como lo es la incorporación de una imagen a un texto predeterminado, etc.

**R**

**Requerimiento:** Funcionalidad requerida por un usuario para resolver un problema o satisfacer uno o varios objetivos.

**T**

**Técnica:** Conjunto de saberes prácticos o procedimientos para obtener un resultado. Requiere de destreza manual e intelectual, y generalmente con el uso de herramientas. Las técnicas se transmiten de generación en generación. La técnica nace en la imaginación y luego se llevan a la concreción, siempre de forma empírica. En cambio, la tecnología surge de forma científica, reflexiva y con ayuda de las técnicas.

**Tecnología:** Abarca un conjunto de técnicas, conocimientos y procesos para el diseño y construcción de objetos para satisfacer necesidades humanas. La tecnología puede referirse a objetos que usa la humanidad (como máquinas, utensilios, hardware), pero también abarca sistemas, métodos de organización y técnicas, se basa en aportes científicos.

**U**

**Usabilidad:** La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y de ser atractivo para el usuario, en condiciones específicas de uso. La usabilidad también hacer referencia al grado de facilidad con que una aplicación, sitio web, periférico o sistema, se adapta a sus usuarios. La usabilidad puede estar relacionada incluso a la Ergonomía, y a la potabilidad de un dispositivo.

**Usuario:** Persona que utiliza el software.

# ANEXOS

## Anexo 1: Manual del archivo .scr

Para poder enviar los comandos APDU se requiere de un formato el cual se explica a continuación:

- ✓ Los comandos tienen que terminar en punto y coma (;).
- ✓ Los comentarios pueden ser (/ /, / * o / **).
- ✓ Se sigue esta sintaxis:
- ✓ CLA INS P1 P2 LC [byte 0 byte 1… byte de LC-1] LE;

  Donde:

  CLA - ISO 7816-4 byte de clase.

  INS - ISO 7816-4 byte de instrucción.

  P1 - ISO 7816-4 byte de parámetro P1.

  P2 - ISO 7816-4 byte de parámetro P2.

  LC - ISO 7816-4 número de bytes de entrada. 1 byte en modo de no extendida, 2 bytes de modo extendido.

LE - ISO 7816-4 longitud de salida prevista. 1 byte en modo de no extendida, 2 bytes de modo extendido.

- ✓ Se escribe primera mente el comando powerup;-> Envía un comando powerup para el lector en la interfaz activa.
- ✓ Para terminar se escribe el comando powerdown;-> Envía un comando powerdown al lector en la interfaz activa.
- ✓ Se ejecuta la herramienta y se obtienen los resultados.

## Anexo 2: Encuesta para Validar Procedimiento

De antemano le agradecemos su colaboración y el tiempo que ha dedicado a la misma.
Marque con una x en la casilla que corresponda a su conocimiento.

1. Su conocimiento acerca de la calidad de software lo puede catalogar como:

☐Deficiente ☐Regular ☐Aceptable ☐Muy Buena ☐Excelente

2. Un elemento crítico para determinar la calidad del software son las pruebas que se le realizan al mismo. ¿Cuánto conoce Ud. acerca del tema?

☐Deficiente ☐Regular ☐Aceptable ☐Muy Buena ☐Excelente

3. En caso de que considere que no sea adecuado el procedimiento de pruebas, ¿Cuáles son, a su entender, las causas por la que esto ocurre?

☐Falta de una adecuada estructura organizativa del procedimiento.

☐Desconocimiento del tema o técnicas a aplicar.

☐Escaso personal capacitado y disponible para trabajar en el procedimiento.

☐Otras_____

_____

4. ¿Considera usted que el procedimiento propuesto está a la altura de las necesidades de un proceso de pruebas para el desarrollo de aplicaciones en tarjetas inteligentes?

☐Si ☐No ¿Por qué?

5. Valorar la propuesta según ciertos criterios establecidos (criterios de mérito científico, implantación, generalización e impacto) dándole valores en una escala del 1 al 5, donde 5 es la máxima evaluación. Los criterios establecidos son:

Satisfacción a las necesidades de equipos de desarrollo ☐1☐2☐3☐4☐5

Calidad de la investigación ☐1 ☐2 ☐3 ☐4 ☐5

Novedad científica ☐1☐2☐3☐4☐5

Aporte científico ☐1☐2☐3☐4☐5

## ANEXOS

Facilidad de comprensión ☐1☐2☐3☐4☐5

Adaptabilidad del procedimiento hacia otros entornos de producción de software ☐1☐2☐3☐4☐5

Repercusión en los proyectos productivos en el desarrollo de las aplicaciones en tarjetas inteligentes ☐1☐2☐3☐4☐5

Contribución al proceso de desarrollo de software ☐1☐2☐3☐4☐5

6. Exprese brevemente algunas sugerencias que usted tendría en cuenta para lograr mejorar la calidad del procedimiento presentado.

_____

_____

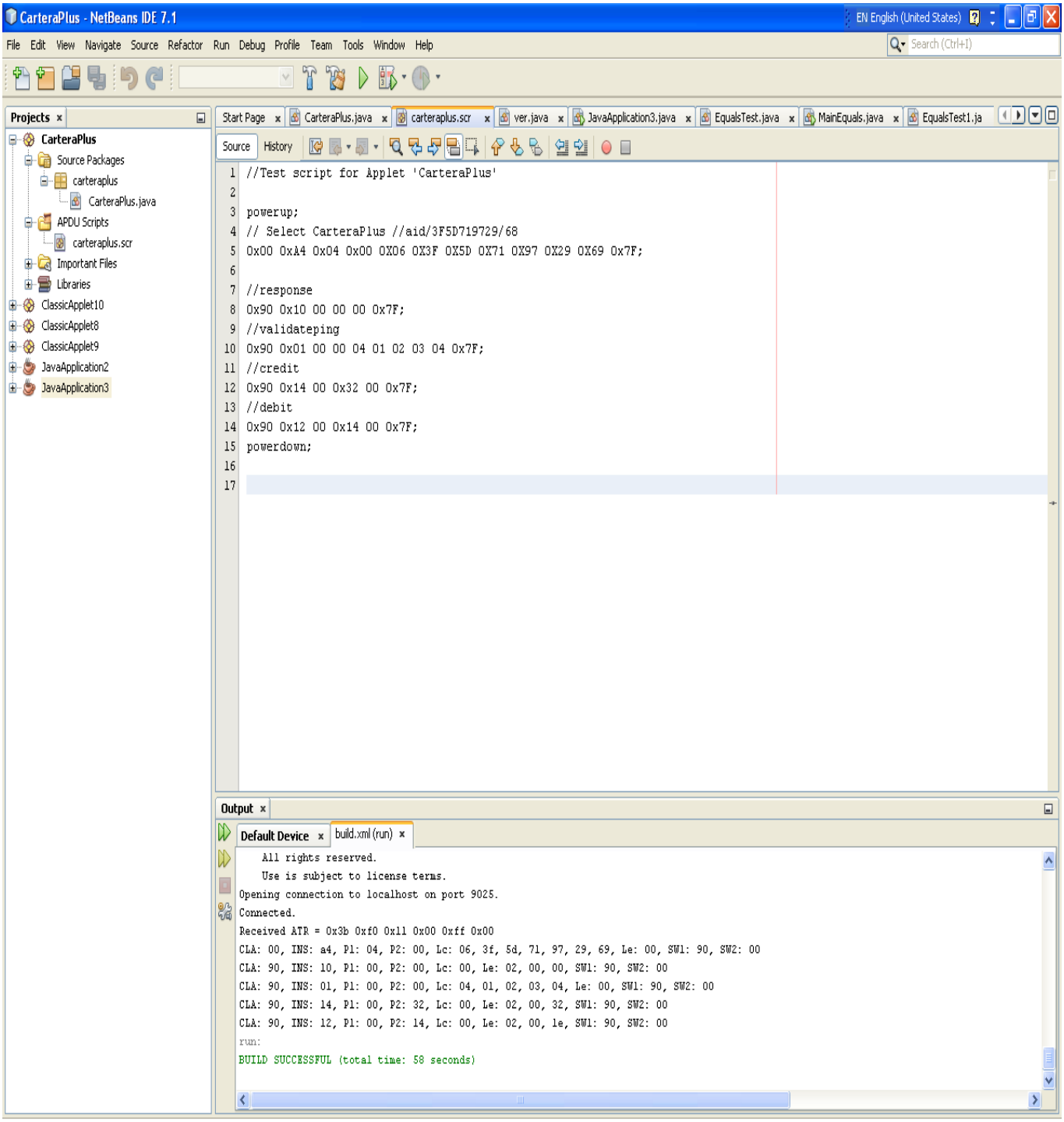
_____

_____

_____

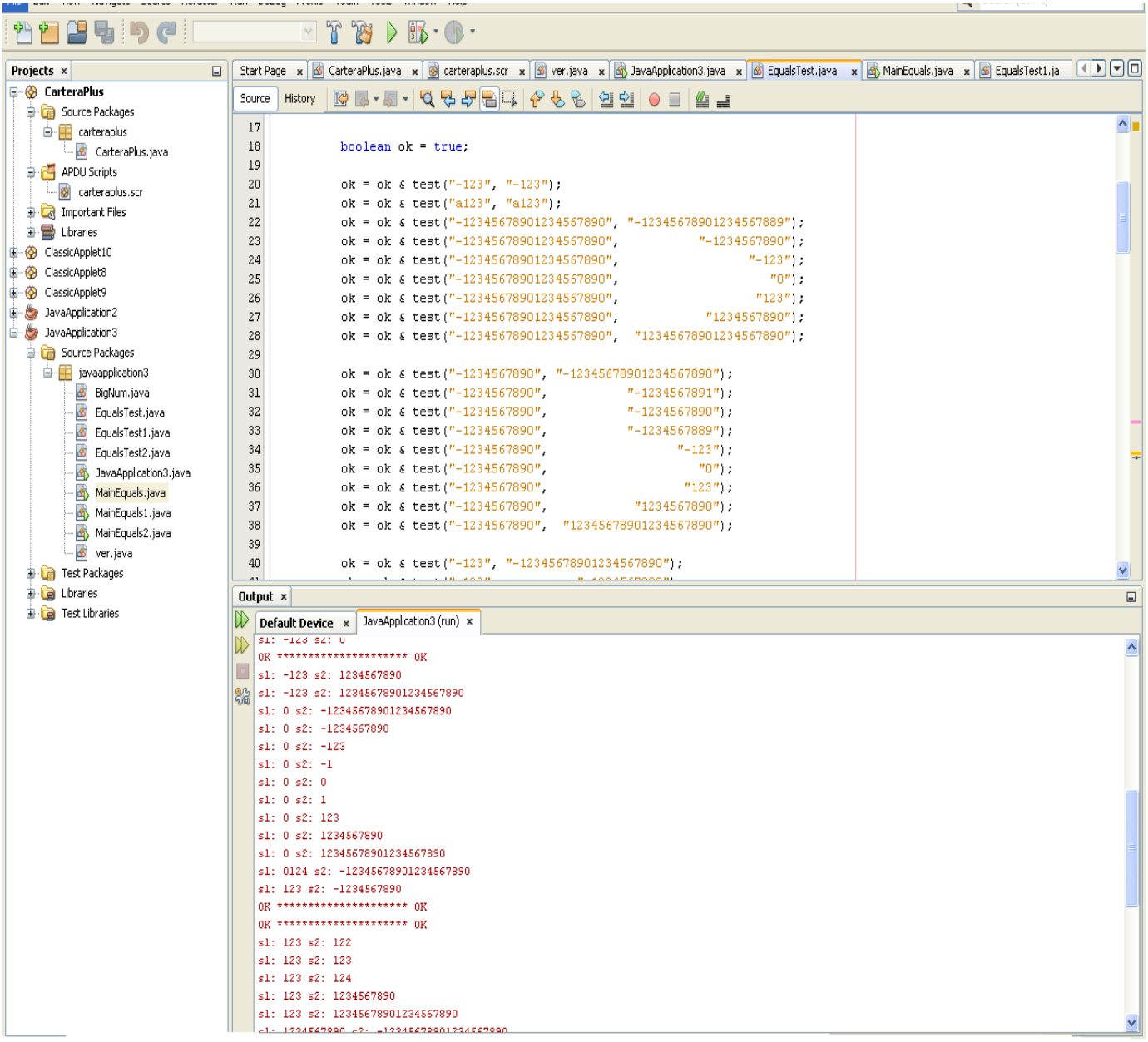
# ANEXOS

## Anexo 3: Pruebas de caja negra



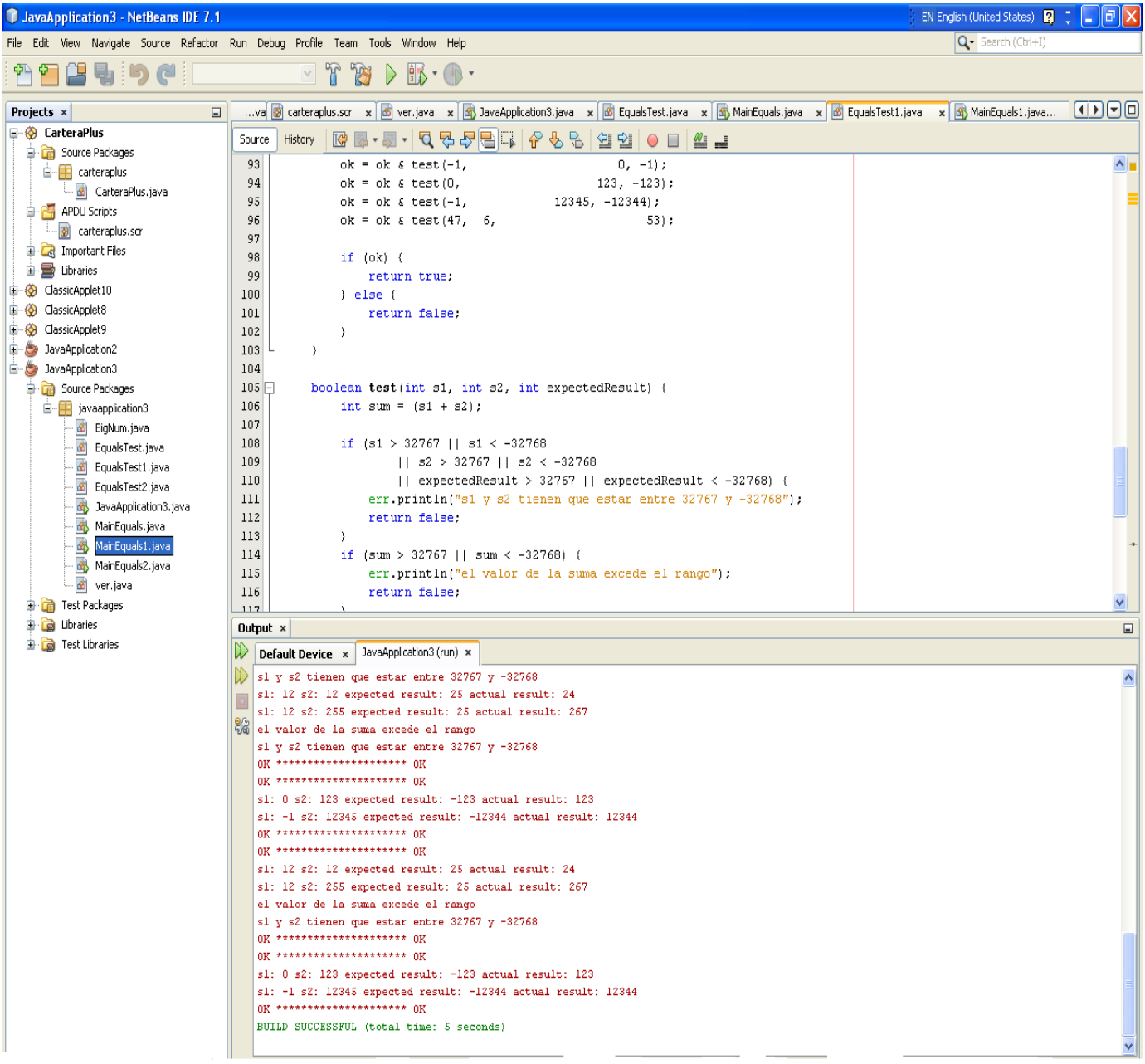
# ANEXOS

## Anexo 4: Pruebas de unidad



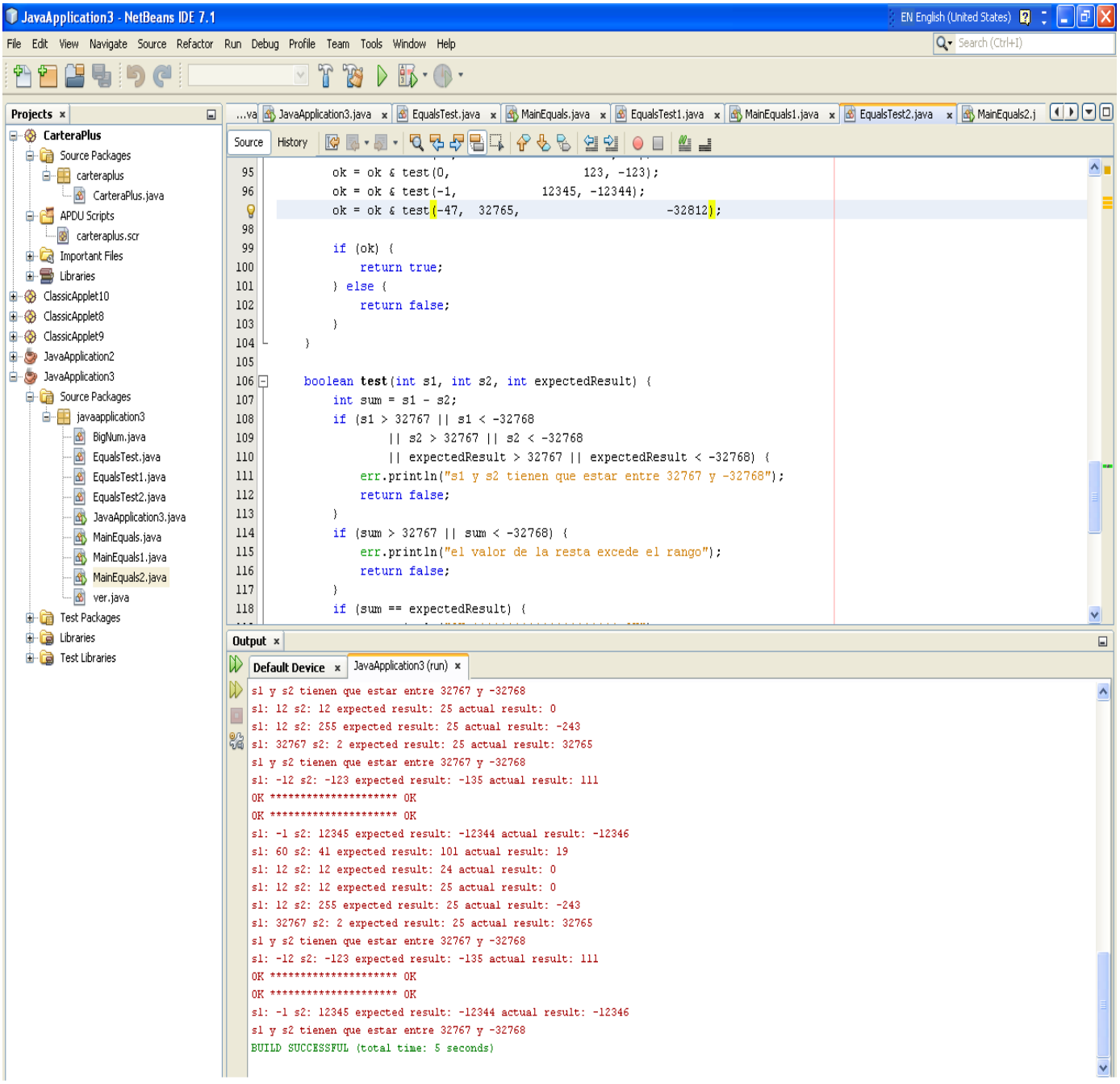
# ANEXOS

## Anexo 5: Pruebas de unidad



# ANEXOS

## Anexo 6: Pruebas de unidad



# ANEXOS

## Anexo 7: Pruebas de unidad