INGENIERIA DE SOFTWARE

Ing. Iván de León

Universidad Mariano Gálvez De Guatemala
Sede San Pedro Sacatepéquez, San Marcos.
Ingeniería En Sistemas Ciencias de la Comunicación.

Ingeniero: Iván Antonio de León.
Curso: Ingeniería de Software.

Estudiante:
María José de León Fuentes
Miguel de Jesus Gonzalez Merida
Claver Mijail Velasquez
Tania Tirado Fuentes

No. Carne:
0903-10-5603
0903-08-4011
0903-10-8106
0903-

Pruebas de Calidad:

El objetivo de las pruebas es presentar información sobre la calidad del producto a las personas responsables de este teniendo esta afirmación en mente, la información que puede ser requerida es de lo más variada. Esto hace que el proceso de testing sea completamente dependiente del contexto en el que se desarrolla. A pesar de lo que muchos promueven, no existen las "mejores prácticas" como tal. Toda práctica puede ser ideal para una situación pero completamente inútil o incluso perjudicial en otra.

Por esto, las actividades, técnicas, documentación, enfoques y demás elementos que condicionarán las pruebas a realizar, deben ser seleccionados y utilizados de la manera más eficiente según contexto del proyecto.

Pruebas estáticas:

Son el tipo de pruebas que se realizan sin ejecutar el código de la aplicación. Puede referirse a la revisión de documentos, ya que no se hace una ejecución de código. Esto se debe a que se pueden realizar "pruebas de escritorio" con el objetivo de seguir los flujos de la aplicación.

Pruebas Dinámicas:

Todas aquellas pruebas que para su ejecución requieren la ejecución de la aplicación. Las pruebas dinámicas permiten el uso de técnicas de caja negra y caja blanca con mayor amplitud. Debido a la naturaleza dinámica de la ejecución de pruebas es posible medir con mayor precisión el comportamiento de la aplicación desarrollada.

Tipos de Pruebas

Las pruebas en conjunto tienen como objetivo general verificar y validar un software, independientemente de las características y el entorno donde se desarrollen, además de los recursos y los factores vinculados al proceso de desarrollo. Hay todo tipo de pruebas, pero nos centraremos en tres de ellas:

Pruebas de Compatibilidad:

Se comprueba el funcionamiento del software desarrollado en muchas plataformas: sistemas operativos, navegadores, redes, hardware...entre otros

Pruebas de Regresión:

Se evalúa el correcto funcionamiento del software desarrollado frente a evoluciones o cambios funcionales.

Pruebas de Integración:

Se centra principalmente en las comunicaciones y las conexiones entre los diferentes módulos del software desarrollado o con terceros (Publicidad, pasarelas de pago, etc.)

Funcionalidad:

- **Función**: Pruebas fijando su atención en la validación de las funciones, métodos, servicios, caso de uso.
- **Seguridad**: Asegurar que los datos o el sistema solamente es accedido por los actores deseados.
- **Volumen**: Enfocada en verificando las habilidades de los programas para manejar grandes cantidades de datos, tanto como entrada, salida o residente en la BD.

Usabilidad:

Prueba enfocada a factores humanos, estéticos, consistencia en la interfaz de usuario, ayuda sensitiva al contexto y en línea, asistente documentación de usuarios y materiales de entrenamiento.

Fiabilidad:

- **Integridad**: Enfocada a la valoración exhaustiva de la robustez (resistencia a fallos).
- **Estructura**: Enfocada a la valoración a la adherencia a su diseño y formación. Este tipo de prueba es hecho a las aplicaciones Web asegurando que todos los enlaces están conectados, el contenido deseado es mostrado y no hay contenido huérfano.
- **Stress**: Enfocada a evaluar cómo el sistema responde bajo condiciones anormales. (extrema sobrecarga, insuficiente memoria, servicios y hardware no disponible, recursos compartidos no disponible).

Rendimiento:

- **Benchmark**: Es un tipo de prueba que compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.
- **Contención**: Enfocada a la validación de las habilidades del elemento a probar para manejar aceptablemente la demanda de múltiples actores sobre un mismo recurso (registro de recursos, memoria).
- **Carga:** Usada para validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante. La variación en carga es simular la carga de

trabajo promedio y con picos que ocurre dentro de tolerancias operacionales normales.

Soportabilidad:

- **Configuración**: Enfocada a asegurar que funciona en diferentes configuraciones de hardware y software. Esta prueba es implementada también como prueba de rendimiento del sistema.
- **Instalación**: Enfocada a asegurar la instalación en diferentes configuraciones de hardware y software bajo diferentes condiciones (insuficiente espacio en disco, etc.)

Tipos de Pruebas por su ejecución:

- Pruebas Manuales
- Pruebas Automáticas

Niveles de Pruebas

- Pruebas Unitarias
- Pruebas de Integración
- Pruebas de Sistema
- Pruebas de Aceptación

El proceso para alcanzar niveles de rendimiento sin precedente.
Pasos

1. Probar la necesidad de mejora.
2. Identificar los proyectos concretos de mejora.
3. Organizar para la conducción de los proyectos.
4. Organizar para el diagnóstico o descubrimiento de las causas.
5. Diagnosticar las causas.
6. Probar que la solución es efectiva bajo condiciones de operación.
7. Proveer un sistema de control para mantener lo ganado.

Diseño de Estándares

En la medida en que se cambian los diferentes grados de abstracción se trabaja para crear abstracciones procedimentales y de datos.

- **Abstracción Procedimental:** Se refiere a una secuencia de instrucciones que tiene una función específica y limitada.

- **Abstracción de Datos:** Es una colección nombrada de datos que describe un objeto de datos.

- Acoplamiento y Cohesión.

- **Acoplamiento de datos:** está subordinado al módulo y se accede a él por medio de una lista convencional de argumentos a través de la cual se pasan los datos.
- **Acoplamiento de marca:** cuando en vez de argumentos simples se pasa una porción de la estructura de datos se pasa por la interfaz del módulo.
- **Acoplamiento de control:** se pasa un indicador de control (una variable que controla las decisiones en el módulo subordinado).
- **Acoplamiento externo:** cuando los módulos están atados a un entorno externo al software. Por ejemplo, las I/O y los dispositivos.
- **Acoplamiento común:** varios módulos hacen referencia a un área global de datos.
- **Acoplamiento de contenido:** un módulo hace uso de datos o de información de control mantenidos dentro de los límites de otro módulo. Cuando se realiza una bifurcación hacia la mitad de otro módulo.

Una clase de diseño cohesiva tiene un conjunto de responsabilidades pequeño y enfocado, y aplica atributos y métodos de manera sencilla de implementar dichas responsabilidades.

Siempre debemos buscar la cohesión más alta, aunque la parte media del espectro es a menudo aceptable.

Coincidencialmente cohesivo: un módulo que realiza un conjunto de tareas poco relacionadas las unas con las otras.

- **Cohesión lógica:** realiza tareas relacionadas lógicamente (produce todas las salidas).
- **Cohesión temporal:** contienen tareas relacionadas por el hecho de que todas deben hacerse en el mismo intervalo de tiempo.
- **Cohesión procedimental:** cuando los elementos de procesamiento están relacionados y deben ejecutarse en un orden específico.
- **Cohesión de comunicación:** todos los elementos de procesamiento se concentran en un área de la estructura de datos.

- La descomposición y la modularización.

Modularidad: Es el atributo particular del software que permite que un programa sea manejable de manera intelectual.

Se divide el software en componentes identificables y tratables por separado, denominados módulos, que están integrados para satisfacer los requisitos del programa.

Hay un número m de módulos que resultarían en un costo de desarrollo mínimo, pero no tenemos la sofisticación necesaria para predecir m con seguridad

- Encapsulación/Ocultar Información

Mediante la agrupación y empaquetado de los elementos y los detalles internos de una abstracción, haciendo que estos detalles sean inaccesibles.

- Separación de la interfaz y la aplicación

La separación de la interfaz y la aplicación implica la definición de un elemento especificando una interfaz pública, conoce a los clientes, aparte de los detalles de cómo se realiza el componente.

- Suficiencia, integridad y primitivismo.

Los métodos asociados con una clase de diseño deben enfocarse en el cumplimiento de un servicio para la clase

Esto es tan fundamental que en todo el proceso de diseño que se debe abordar de una manera u otra:

- Concurrencia:

La forma de descomponer el software en los procesos, tareas e hilos tratar relacionarlos con la eficiencia, la atomicidad, la sincronización, y demás cuestiones de programación.

- Control y manejo de Eventos

Cómo organizar los datos y el controlar el flujo, manejo de reactivo y temporal de los acontecimientos a través de diversos mecanismos, tales como la invocación implícita de llamadas y sus intentos.

- Distribución de Componentes

Cómo distribuir el software en el hardware, cómo los componentes se comunican, cómo se puede usar una plataforma al utilizarse para hacer frente a software heterogéneos.

- Error y Gestión de Excepciones Tolerancia a Fallos.

El análisis y la gestión del riesgo son una serie de pasos que ayudan al equipo del software a comprender y a gestionar la incertidumbre.

Un riesgo es un problema potencial que puede ocurrir o no. Pero sin tener en cuenta el resultado, realmente es una buena idea es identificarlo, evaluar su probabilidad de aparición, estimar.

Calidad en el Análisis, Diseño y Evaluación del Software:

- **Calidad de atributos**

Varios atributos son generalmente considerados importantes que permiten obtener un diseño de software con alta calidad, existen algunas características que son (mantenible, portabilidad, probable) y (correctos, robusto). Cabe destacar que existen diferencias entre calidad de atributos que son (rendimiento, seguridad, funcionalidad y usabilidad), y los que son (portabilidad, reutilización, integralidad y pruebas), y las características relacionadas con la arquitectura (integridad conceptual, correcto, completo).

- **Calidad en análisis y evaluación de técnicas**

Varias técnicas y herramientas pueden ayudar a mejorar la calidad de diseño de software:

- Diseño de software: Para este tipo se puede aplicar al diseño de software informal y semi informal tomando un grupo base, técnicas que permiten verificar la calidad de diseño de los artefactos que pueden ser (vista de la arquitectura, diseño -inspección, técnicas y requerimientos).

- Análisis estático: Para este tipo se puede aplicar al diseño de software informal y semi informal que permite evaluar algo simple utilizando análisis automáticos de casos de pruebas.

Herramientas

(gsBase(ie))

Es una herramienta de desarrollo y explotación de propósito general que se puede utilizar para crear soluciones para ingeniería, cálculo, gestión, diseño gráfico, educación, etc. Incluye novedosas ideas originales que le harán su trabajo mucho más fácil. Permite construir soluciones integrales para empresas (ERP's, CRM's, B2B, B2C, ...) minimizando el tiempo invertido, recursos hardware y costes de desarrollo.
La comunicación entre cliente y servidor se puede establecer por redes locales internas, Internet o redes privadas. Los recursos mínimos de comunicaciones necesarios hacen posible usar distintas tecnologías de comunicación: ADSL, GPRS, UMTS, RTB, Frame Relay, 3G, RDSI, etc.

DIAGRAMAS ANALISIS DE DISEÑO

DIAGRAMA DE CASO DE USO

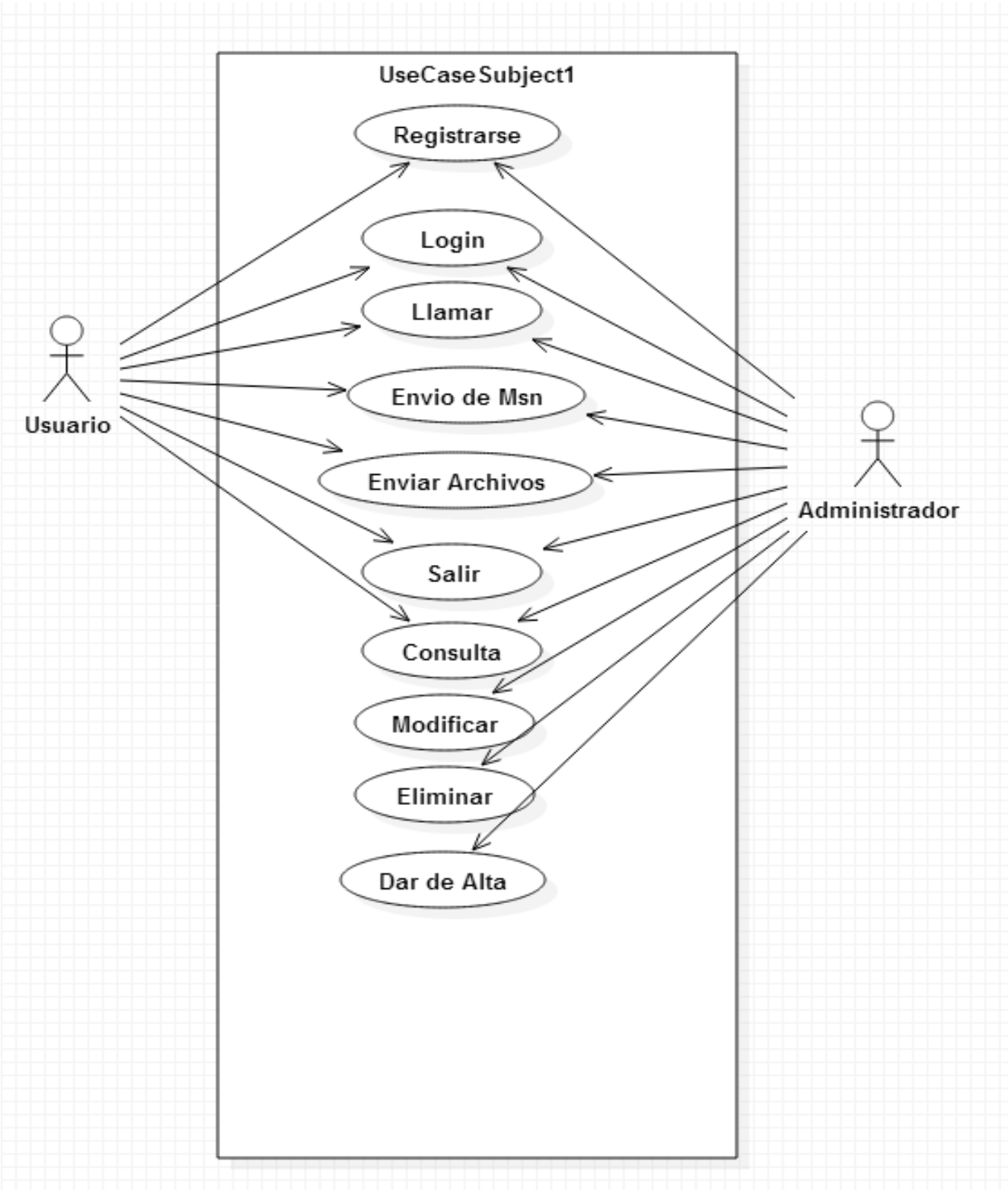


DIAGRAMA DE SECUENCIA ADMINISTRADOR

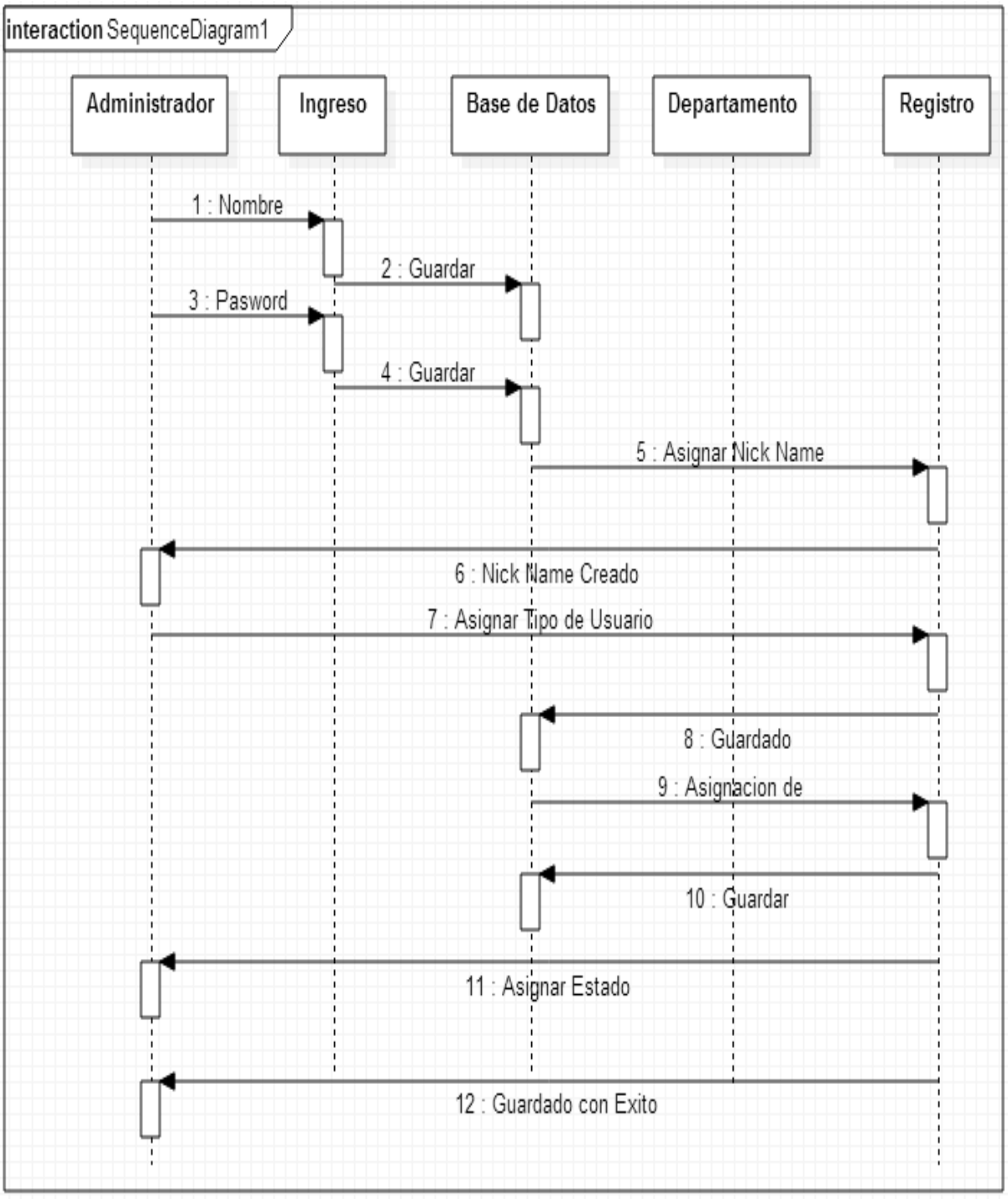


DIAGRAMA DE SECUENCIA USUARIO

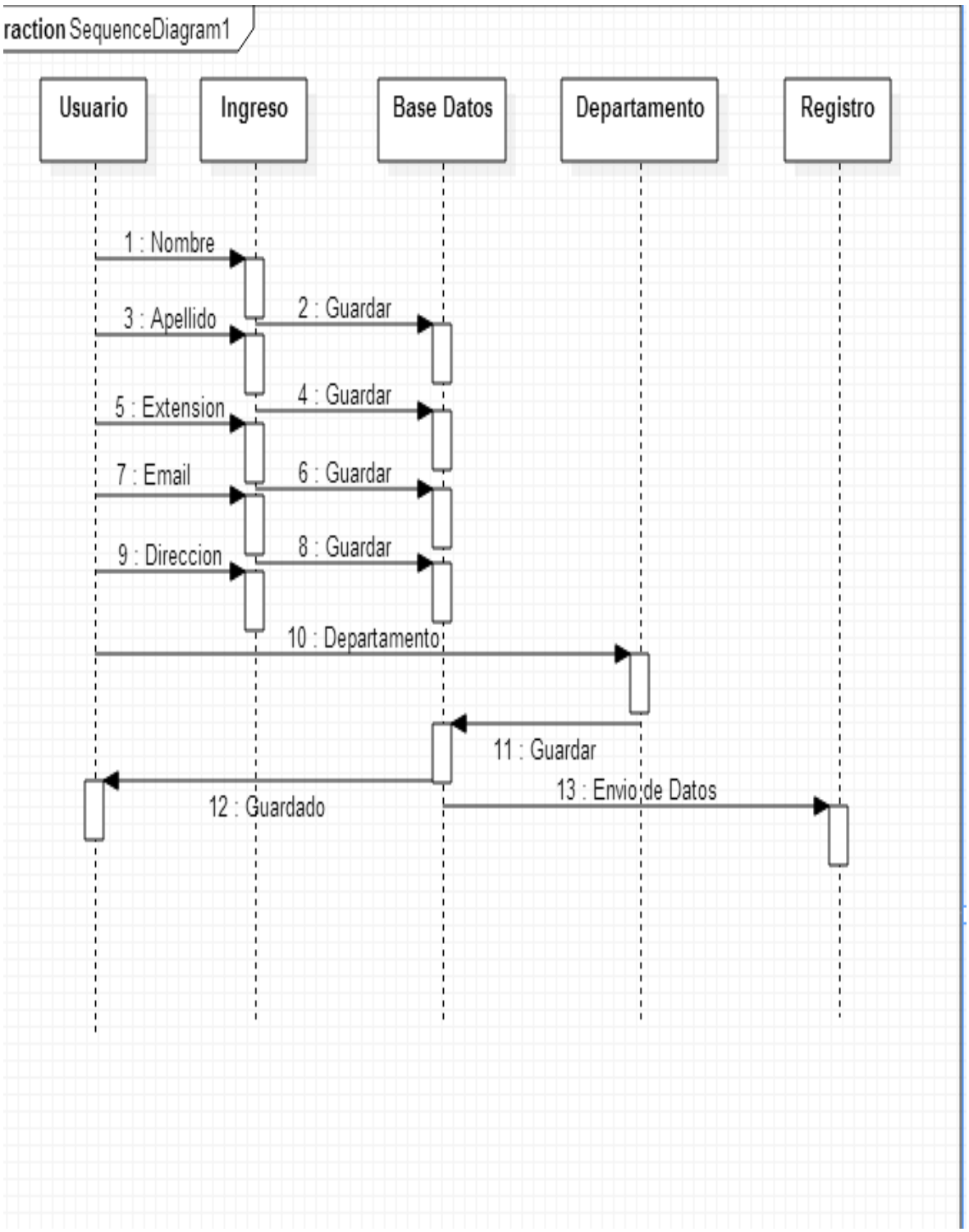
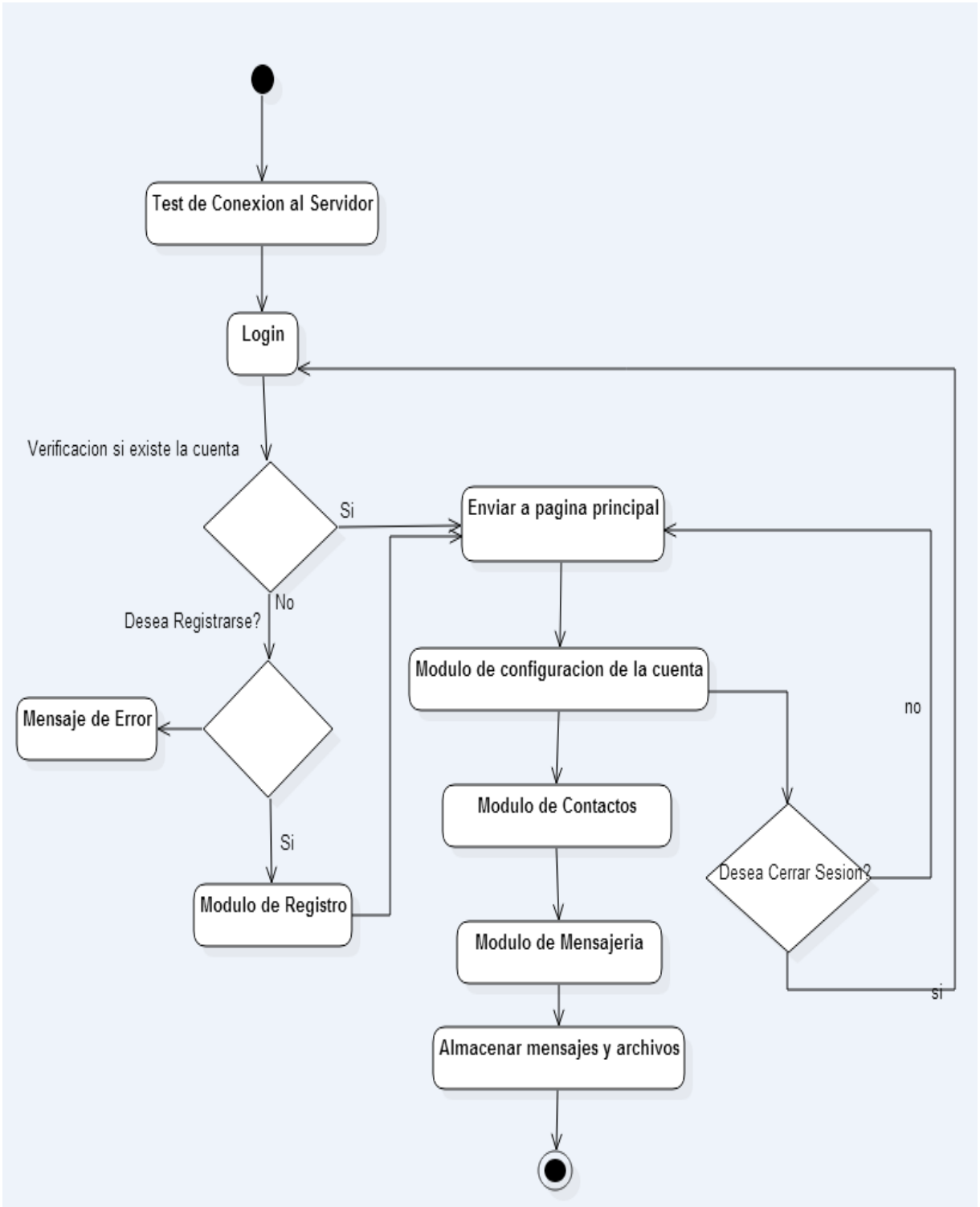


DIAGRAMA DE FLUJO DE PROCESOS



DISEÑO BASE DE DATOS VERSION 1